Vigilada Mineducación

MARCO DE IMPLEMENTACIÓN DE LABORATORIOS DE PROGRAMACIÓN APLICADO A CURSOS INTRODUCTORIOS DE PROGRAMACIÓN PARA ESTUDIANTES UNIVERSITARIOS

FRAMEWORK FOR IMPLEMENTING PROGRAMMING LABORATORIES APPLIED TO INTRODUCTORY PROGRAMMING COURSES FOR UNDERGRADUATE STUDENTS

AGUSTÍN NIETO GARCÍA

Trabajo de grado

Asesores
Daniel Correa Botero
Paola Andrea Vallejo Correa

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería
Maestría en Ingeniería
Medellín
2024

Agradecimientos

Quiero agradecer de corazón a mis asesores por su paciencia y apoyo incondicional. Esta tesis no habría sido posible sin su ayuda constante y su comprensión.

Resumen

En el panorama actual, la programación se vuelve cada vez más relevante, y su enseñanza se ha expandido más allá de las disciplinas informáticas tradicionales. Este crecimiento ha llevado a la inclusión de cursos introductorios de programación en una variedad de programas universitarios, abordando el aumento en la necesidad de habilidades digitales. Aprender a programar implica desarrollar habilidades de resolución de problemas y pensamiento algorítmico, lo que requiere herramientas efectivas y un soporte adecuado para los estudiantes. Sin embargo, la selección de herramientas apropiadas puede ser desafiante debido a la gran cantidad de opciones disponibles. Este estudio se enfoca en analizar y proponer un marco para la implementación de laboratorios de programación en contextos universitarios, con el objetivo de facilitar la elección y aplicación de herramientas de enseñanza más adecuadas.

Palabras clave: programación, educación, herramienta, laboratorio

Abstract

In the current landscape, programming is becoming increasingly relevant, and its teaching has expanded beyond traditional computer science disciplines. This growth has led to the inclusion of introductory programming courses in a variety of university programs, addressing the growing need for digital skills. Learning to program involves developing problem-solving skills and algorithmic thinking, which require effective tools and adequate support for students. However, selecting appropriate tools can be challenging due to the plethora of options available. This study focuses on analyzing and proposing a framework for the implementation of programming labs in university contexts, aiming to facilitate the selection and application of more suitable teaching tools.

Keywords: programming, education, tool, laboratory.

Tabla de contenidos

Lista de Tablas

Lista de Figuras

Capítulo 1: Introducción

En el mundo actual, tener conocimientos generales de programación es cada vez más relevante. Tradicionalmente los cursos de programación se ofrecían en programas asociados a computación y/o informática; sin embargo, durante los últimos años estos cursos han proliferado en múltiples disciplinas y programas [1]. Hoy en día, es común ver cómo diferentes universidades ofrecen cursos introductorios de programación como componente obligatorio para múltiples programas, o incluso para facultades completas [2]. Sin embargo, hay un déficit de habilidades en talento digital, razón por la cual cobra mayor relevancia la formación en programación. Aprender a programar implica aprender a planificar, escribir y ejecutar código en diversos lenguajes de programación. Todo esto con el objetivo de lograr que el estudiante adquiera habilidades de resolución de problemas, desarrollando un pensamiento algorítmico básico. Definir herramientas y soporte adecuado para los estudiantes, resulta crucial para facilitar el proceso de aprendizaje en este tipo de cursos. Cualquier esfuerzo que facilite, agilice y mejore la educación en programación es de gran impacto.

Existen múltiples maneras de diseñar y ejecutar cursos introductorios de programación, estas pueden ir desde el diseño de cursos que no utilizan herramientas de automatización y donde todos los procesos de creación de ejercicios, calificación de actividades y gestión de las notas se realizan de manera manual, hasta cursos donde se utilizan una serie de plataformas que automatizan estos procesos.

Muchos cursos de programación se planifican como MOOCs (cursos en línea masivos de acceso abierto) y generalmente se administran a través de plataformas LMS (sistemas de administración de aprendizaje) [3]. Sin embargo, dependiendo de la planificación del curso y la cantidad de estudiantes, algunas tareas como la administración, calificación de actividades y retroalimentación hacia los estudiantes, se pueden volver muy tediosas. Incluso pueden hacer que los profesores inviertan mucho tiempo en procesos manuales que podrían ser automatizados y que las retroalimentaciones no sean instantáneas, haciendo que los estudiantes no puedan corregir sus errores oportunamente. Debido a este escenario, algunos procesos tales como la calificación automatizada de los ejercicios desarrollados por los estudiantes y la retroalimentación inmediata se están convirtiendo en una necesidad más que en una opción.

Se han llevado a cabo diversas investigaciones sobre herramientas destinadas a mejorar la enseñanza de la programación para principiantes. Entre estas herramientas, se encuentran varias que facilitan la gestión de cursos de programación. Por ejemplo, ofrecen funciones de calificación automatizada, visualización de errores comunes, estadísticas sobre el progreso de los estudiantes y análisis de su frecuencia de conexión para trabajar, así como del tiempo dedicado al estudio, entre otras características. También hay otras herramientas que ayudan a los estudiantes a comprender conceptos, como los lenguajes de programación visual, las herramientas que les permiten seguir paso a paso la ejecución de los programas (como los *debuggers*), y las herramientas diseñadas para ofrecer pistas a los estudiantes si las solicitan o si consideran que llevan mucho tiempo atascados, entre otras opciones disponibles.

En la actualidad, resulta desafiante seleccionar las herramientas más adecuadas para contextos específicos de enseñanza debido a la amplia variedad disponible. Esta dificultad surge porque existen numerosas herramientas diseñadas para diferentes contextos, y no todas son igualmente

aplicables a un caso particular. Esto dificulta la tarea de seleccionar, probar y finalmente implementar dichas herramientas. Diferentes autores han propuesto herramientas y realizado estudios sobre el uso de éstas, sin embargo, estos trabajos se suelen concentrar en actividades de desarrollo y prueba de herramientas específicas o en la comparación entre estas. Generalmente se omite la evaluación directa de herramientas, así como el análisis de los beneficios y desventajas de su implementación, y la provisión de pautas para seleccionar la más adecuada según contextos específicos. Esto conlleva a que las herramientas mencionadas en la literatura y en revisiones literarias estén frecuentemente desactualizadas o abandonadas, y en la mayoría de los casos no estén disponibles para uso público, lo que dificulta su utilización por parte de terceros.

El objetivo principal de este trabajo es analizar las herramientas para la enseñanza de programación básica en universidades e identificar diferentes contextos que se puedan encontrar en diferentes cursos para definir una metodología de implementación de laboratorios de programación que aplique a esos contextos. Proponemos una aplicación que permita seleccionar las herramientas que mejor se adecúen a los contextos específicos, como pueden ser limitaciones de presupuesto, número de estudiantes, selección de lenguaje de programación, necesidad de visualización de los algoritmos, calificación automatizada, disponibilidad en línea, etc. Dicha herramienta tiene como objetivo facilitar la tarea de hallar las herramientas de enseñanza que mejor se ajusten a un contexto o necesidades específicas.

El documento está estructurado de la siguiente manera: El capítulo 1 presenta las motivaciones y la relevancia de esta investigación. El capítulo 2 se concentra en presentar los antecedentes y conceptos necesarios para comprender el desarrollo de la investigación. El capítulo 3 presenta la metodología seguida durante esta investigación. El capítulo 4 presenta un mapeo de la literatura sobre herramientas destinadas al apoyo de la programación introductoria. El capítulo 5 describe la propuesta del marco para la implementación de laboratorios de programación en cursos universitarios introductorios. El capítulo 6 muestra la aplicación de dicho marco en un contexto específico. El capítulo 7 presenta las conclusiones y el trabajo futuro de este estudio.

Capítulo 2: Marco teórico

A continuación, se listan definiciones de conceptos que serán importantes para entender el contenido de este documento.

Algoritmo: Un algoritmo es "un conjunto finito de instrucciones o reglas bien definidas y ordenadas que describe un proceso o conjunto de operaciones que realiza una tarea o resuelve un problema específico" [4]. Un algoritmo toma una entrada, ejecuta una serie de pasos u operaciones según reglas definidas, y produce una salida que resuelve un problema o realiza una tarea específica. Los algoritmos se emplean en campos diversos como la informática, las matemáticas, la ingeniería y muchas otras disciplinas. La comprensión de la algoritmia es fundamental en los cursos introductorios universitarios de programación para desarrollar habilidades en esta área.

Programación: La programación es "el acto de programar, es decir, organizar una secuencia de pasos ordenados a seguir para hacer cierta cosa. Este término puede utilizarse en muchos contextos, es común hablar de programación a la hora de organizar una salida, las vacaciones o de la lista de programas con sus días y horarios de emisión de los canales de televisión o la lista de películas de un cine" [5]. Al entender cómo organizar y estructurar pasos para resolver problemas, los estudiantes adquieren habilidades fundamentales para programar de manera efectiva. Esta comprensión les permite descomponer problemas complejos en pasos más simples y abordarlos de manera sistemática. Además, al practicar la creación de algoritmos y su implementación en código, los estudiantes desarrollan su capacidad para pensar de manera lógica y para resolver problemas de manera eficiente, habilidades esenciales en el campo de la programación y en muchas otras áreas de la informática y la ingeniería.

Lenguaje de programación: Un lenguaje de programación es una "estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora" [6]. En cursos de programación, los estudiantes no solo aprenden a desarrollar algoritmos, sino también a expresar esos algoritmos en un lenguaje de programación específico. Los lenguajes de programación actúan como herramientas para traducir la lógica y la secuencia de pasos de un algoritmo en un conjunto de instrucciones comprensibles por la computadora. Por lo tanto, al aprender un lenguaje de programación, los estudiantes adquieren la capacidad de comunicarse eficazmente con las computadoras y de implementar soluciones prácticas a problemas utilizando la potencia de la tecnología. Esto les proporciona una base sólida para su futuro en el campo de la informática y les capacita para desarrollar software funcional.

MOOC (Massive Open Online Course - curso en línea masivo y abierto): Los MOOC son una de las tendencias más destacadas de la enseñanza superior en los últimos años. El término MOOC se refiere a contenidos didácticos, vídeos, conjuntos de problemas y foros de libre acceso, globales y gratuitos, difundidos a través de una plataforma en línea a un gran número de participantes que desean seguir un curso o recibir formación. Los MOOC suelen ofrecerse a través de una plataforma en línea que permite el acceso a un gran número de participantes que desean seguir un curso o formarse [3]. Muchas instituciones educativas y plataformas en línea ofrecen MOOC especializados en programación, que abarcan una amplia gama de temas, desde conceptos básicos hasta temas más avanzados. Estos cursos brindan a los estudiantes la oportunidad de aprender a programar desde cualquier lugar y en cualquier momento, lo que los hace especialmente atractivos

para aquellos que desean adquirir habilidades en programación, pero no tienen acceso a programas educativos tradicionales.

LMS (Learning Management System - sistema de gestión del aprendizaje): Una plataforma LMS (Learning Management System) es un espacio virtual diseñado para mejorar la experiencia del aprendizaje a distancia. Los LMS permiten crear aulas virtuales para impartir clases en línea, organizar materiales y actividades de formación en cursos, y gestionar las matrículas de los estudiantes. Además, facilita el seguimiento de los procesos de aprendizaje y evaluación. También ofrece herramientas de comunicación como foros y chats, a través de los cuales los estudiantes pueden resolver dudas, realizar exámenes, intercambiar información, entre muchas otras opciones [7].

Herramienta para la enseñanza de programación: Se refiere a cualquier software, aplicación, recurso en línea o dispositivo físico diseñado para facilitar el proceso de enseñanza y aprendizaje de la programación. Estas herramientas están diseñadas para ayudar a los estudiantes a comprender los conceptos fundamentales de la programación, practicar la escritura de código, depurar programas y desarrollar habilidades de resolución de problemas algorítmicos. Entre las herramientas para la enseñanza de programación se incluyen entornos de desarrollo integrados, simuladores de código, plataformas de aprendizaje en línea, juegos educativos, tutoriales interactivos, entre otros recursos. La variedad de herramientas disponibles permite a educadores y estudiantes elegir aquellas que mejor se adapten a sus necesidades, estilos de aprendizaje y objetivos educativos específicos.

Estrategia para la enseñanza de programación: Plan o enfoque diseñado para facilitar el aprendizaje efectivo de habilidades de programación. Se busca proporcionar a los estudiantes un marco estructurado para adquirir conocimientos y habilidades de manera progresiva y comprensible. Una estrategia debe incluir: metodología (*pair programming*, *live coding*, etc.), contextos a los que aplica (juegos, uso de hardware, etc.), herramientas a utilizar (lenguaje de programación, LMS, etc.) y otros aspectos como pueden ser resolución de dudas, manejo del lenguaje, entre otros [8].

Laboratorio de programación: Entorno físico equipado con recursos y herramientas destinados a facilitar la práctica y el aprendizaje de la programación informática [9]. En un laboratorio de programación, los estudiantes tienen acceso a computadoras, software especializado, material didáctico y asistencia técnica para llevar a cabo ejercicios prácticos, proyectos y actividades relacionadas con la programación. Estos laboratorios pueden estar ubicados en instituciones educativas, como universidades, colegios y centros de formación profesional, o pueden ser entornos virtuales accesibles a través de internet. El propósito principal de un laboratorio de programación es proporcionar un espacio donde los estudiantes puedan aplicar los conocimientos teóricos adquiridos en clases y cursos de programación, experimentar con diferentes lenguajes y tecnologías de programación, resolver problemas prácticos y desarrollar habilidades prácticas en el desarrollo de software.

Laboratorio virtual de programación: Herramienta para enseñanza de la programación (normalmente aplicación web) que habilita a los usuarios a realizar laboratorios de programación de manera virtual. Ofrece funciones de edición, ejecución y evaluación automatizada de programas

que facilitan el proceso de aprendizaje de los alumnos y la tarea de evaluación de los profesores [10]. El mayor exponente de este tipo de herramientas para la educación es MoodleVPL [11].

Capítulo 3: Metodología

Para llevar a cabo el presente trabajo se utilizó una estrategia de investigación basada en la metodología de la ciencia del diseño (o *design science*) propuesta por Peffers *et al.* [12].

La Figura 1 presenta la metodología utilizada en esta investigación, basada en seis etapas fundamentales: (i) identificación y motivación del problema, donde se define el problema a resolver y su relevancia; (ii) definición de objetivos, donde se establecen los criterios y metas para la solución; (iii) diseño y desarrollo, donde se lleva a cabo la creación del artefacto destinado a solucionar el problema identificado; (iv) demostración, donde se aplica el artefacto propuesto en un contexto específico para validar su funcionalidad; (v) evaluación, donde se analiza el desempeño del artefacto respecto a los objetivos planteados; y (vi) comunicación, donde se difunden los resultados y hallazgos de la investigación. Cada etapa se explica con mayor detalle a continuación:

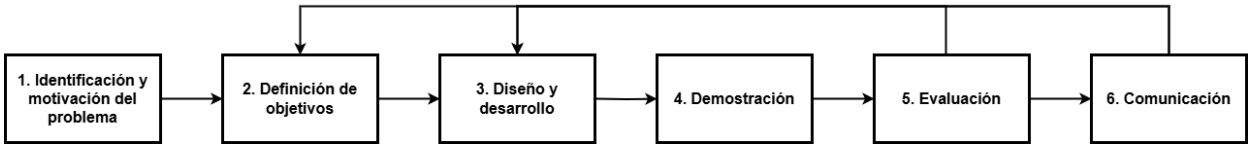

Figura 1. Metodología de la ciencia del diseño

Identificación y motivación del problema

Este trabajo se centró en abordar la problemática de cómo implementar laboratorios de programación aplicados a cursos introductorios de programación para estudiantes universitarios. Para abordar esta problemática, se realizó un mapeo de literatura que abarcó investigaciones preliminares publicadas por distintos autores entre los años 2012 y 2023. Los hallazgos de este estudio se presentan detalladamente en el Capítulo 4.

Definición de objetivos

El objetivo general del presente trabajo es *"desarrollar un marco de implementación de laboratorios de programación aplicado a cursos universitarios introductorios de programación"*.

La Tabla 1 lista los objetivos de investigación específicos del presente trabajo.

ID	Objetivo
O1	Identificar en la literatura las herramientas actuales de implementación de laboratorios de programación.
O2	Analizar y definir las ventajas y desventajas de las herramientas encontradas en la literatura.
O3	Identificar elementos de contexto, que sean relevantes para la definición de una implementación de laboratorio de programación aplicado a cursos universitarios introductorios de programación.
O4	Proponer un marco de implementación de laboratorios de programación que aplique a diferentes contextos de cursos universitarios introductorios de programación.
O5	Implementar y evaluar la propuesta en un curso universitario.

Tabla 1. Objetivos específicos de la investigación

Diseño y desarrollo

En la etapa de diseño y desarrollo se definió un marco de implementación de laboratorios de programación aplicado a cursos universitarios introductorios de programación. Los detalles de este marco se presentan en el Capítulo 5.

Demostración

En la etapa de demostración se ejecutó el marco de implementación de laboratorios de programación y se aplicó a un caso de estudio de un curso introductorio de programación de la universidad EAFIT. Esta etapa se detalla en el Capítulo 6.

Evaluación

Una vez aplicado el marco de implementación de laboratorios de programación al curso introductorio de programación de la universidad de EAFIT, se evalúa la utilidad y el alcance del marco propuesto. Esta etapa se detalla en el Capítulo 7.

Comunicación

Finalmente, para la etapa de comunicación se propone el presente documento cuyo objetivo es el de comunicar el desarrollo de la propuesta y los resultados obtenidos.

Capítulo 4: Mapeo de literatura

En el contexto actual, donde la programación se ha vuelto esencial en diversos campos del conocimiento, surgen múltiples enfoques para su enseñanza, los cuales permiten brindar apoyo tanto a estudiantes como a instructores. La creciente demanda de formación en programación ha dado lugar a la aparición de laboratorios virtuales como herramientas innovadoras que facilitan el aprendizaje práctico. Estos entornos permiten a los estudiantes practicar habilidades de programación, mientras que los profesores pueden explorar diversas estrategias y herramientas para impartir cursos prácticos de programación.

Aunque investigaciones previas han aportado valiosas ideas sobre el uso de laboratorios virtuales, se han pasado por alto aspectos críticos como la accesibilidad y el costo de las herramientas. Las soluciones de código abierto podrían resolver este problema al ofrecer opciones accesibles y rentables. Sin embargo, en muchos casos solo se encuentran disponibles herramientas comerciales, lo que plantea limitaciones en términos de accesibilidad y asequibilidad. Además, la investigación previa se ha centrado más en la publicación de herramientas que en su funcionalidad y usabilidad en situaciones prácticas, dejando lagunas críticas en la comprensión de su beneficio real para estudiantes e instructores en la enseñanza de programación. Este capítulo aborda estas deficiencias y explora el uso de dichas herramientas para la enseñanza de la programación introductoria en la educación universitaria, mediante un mapeo sistemático de literatura (SMS, por sus siglas en inglés) que revisa 89 publicaciones desde 2012 hasta 2023. Este SMS sirve como punto de partida esencial para futuras investigaciones, guiando a educadores e investigadores en el aprovechamiento de los laboratorios de programación en la educación universitaria.

La estructura de este capítulo es la siguiente: La sección 4.1 expone la metodología empleada para el mapeo sistemático de literatura, que abarca desde la selección de la cadena de búsqueda hasta la selección de bases de datos y el proceso de extracción de datos. La sección 4.2 ofrece un análisis de las publicaciones seleccionadas y de la información recopilada. La sección 4.3 presenta las discusiones del estudio. La sección 4.4 analiza las posibles amenazas a la validez del estudio. La sección 4.5 presenta las conclusiones del estudio, se abordan los temas abiertos a investigación en este campo y se esbozan oportunidades para futuras investigaciones.

4.1 Metodología

Esta investigación aplicó las etapas de la guía propuesta por Petersen *et al.* (2008) [13] para desarrollar un mapeo sistemático de literatura relacionado con el uso de herramientas en cursos universitarios de programación. En concreto, nuestro proceso de mapeo de la literatura consta de seis etapas, presentadas en la Figura *2*: (i) definición de las preguntas de investigación (véase la subsección 4.1.1), (ii) ejecución de la estrategia de búsqueda (véase la subsección 4.1.2), (iii) selección de publicaciones (véase la subsección 4.1.3), (iv) extracción de datos (véase la subsección 4.1.4), (v) síntesis de datos (véase la subsección 4.1.5) y (vi) informe de resultados (véase la sección 4.2). Cada etapa produce un resultado que se utiliza como entrada para la etapa siguiente.

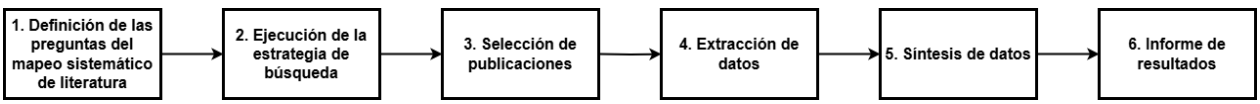

Figura 2. Proceso de mapeo de la literatura

4.1.1 Definición de las preguntas del mapeo sistemático de literatura

El objetivo principal de este capítulo es presentar un SMS sobre el uso de laboratorios virtuales de programación y herramientas similares para la enseñanza de cursos introductorios de programación en la educación universitaria. Esperamos proporcionar evidencias que puedan ayudar a identificar oportunidades de investigación en el tema en cuestión.

Este mapeo de literatura pretende (i) descubrir tendencias en las publicaciones dentro del dominio de las herramientas utilizadas para la enseñanza de la informática para principiantes e (ii) identificar los lenguajes de programación específicos y las herramientas empleadas en la enseñanza de cursos sobre este tema.

La pregunta principal de este mapeo de literatura es: *"¿Cómo se utilizan los laboratorios virtuales de programación y herramientas similares para apoyar la enseñanza de cursos introductorios de programación en instituciones universitarias?"* La Tabla *2* presenta las preguntas de investigación (PI) específicas que proporcionan respuesta a la pregunta principal.

ID	**Descripción**
PI1	¿Qué autores, en qué países y con qué frecuencia se han publicado investigaciones sobre el uso de laboratorios virtuales de programación para enseñar programación introductoria en entornos universitarios?
PI2	¿Qué lenguajes y herramientas de programación se utilizan para dictar cursos de introducción a la programación en instituciones universitarias?
PI3	¿Con qué fines y en qué fases del proceso educativo se utilizan los laboratorios virtuales de programación?
PI4	¿Cuáles son las tendencias y los retos de la utilización de laboratorios virtuales de programación en la educación?

Tabla 2. Preguntas de investigación del mapeo de literatura

4.1.2 Ejecución de la estrategia de búsqueda

En esta sección se presenta el proceso de selección de las publicaciones para el SMS.

Definición de la cadena de búsqueda

Antes de definir la cadena de búsqueda del SMS, realizamos búsquedas preliminares de publicaciones para descartar aquellos términos que generaban resultados irrelevantes de otras áreas del conocimiento, así como los que eran demasiado específicos o, por el contrario, demasiado generales, limitando la efectividad de los resultados. También buscamos hallar los términos que mejor comunicaban nuestra intención de búsqueda a todos los motores y que nos proporcionaban resultados relevantes para nuestra investigación.

Para llevar a cabo estas búsquedas utilizamos las palabras clave y los sinónimos presentados en la Tabla *3*. Los término de búsqueda y la cadena de búsqueda se definieron en inglés debido a que las

búsquedas preliminares revelaron que la mayoría de la literatura relevante sobre el tema se encuentra publicada en este idioma.

Palabra clave	**Sinónimos**
Undergraduate course	undergraduate students, first year undergraduate course, introductory course, university course
Programming laboratory	VPL, virtual programming lab, programming lab, coding lab, coding laboratory, virtual classroom
Programming fundamentals	computer science fundamentals, computational thinking
Learning methodology	Learning strategy, education strategy

Tabla 3. Sinónimos de las palabras claves

Luego de analizar las publicaciones encontradas, definimos la siguiente cadena de búsqueda como base para el desarrollo del SMS: *"programming course" AND "laboratory" AND ("university" OR "higher education") AND "tool"*.

Selección de la base de datos

Seleccionamos las bases de datos basándonos en varios factores, como son: (i) los temas publicados, (ii) la cantidad de publicaciones disponibles y (iii) el prestigio de la base de datos. En principio, se incluyeron Scopus, Web of Science, ScienceDirect, ACM e IEEE por su relevancia y amplitud en investigaciones sobre programación y su enseñanza, asegurando una cobertura exhaustiva y de alta calidad. Google Scholar, por ejemplo, fue excluido debido a su dificultad de acceso a publicaciones, falta de validación y menor reconocimiento académico.

La Tabla *4* muestra las bases de datos consideradas en el estudio y si se incluyeron o descartaron después de realizar búsquedas de prueba. En este caso, sólo se descartó SpringerLink porque sus resultados no estaban relacionados con nuestros temas de interés. Finalmente, las bases de datos seleccionadas para este estudio fueron Scopus, Web of Science, ScienceDirect, ACM e IEEE.

Base de datos	**Seleccionada**
ACM	Sí
IEEE	Sí
ScienceDirect	Sí
Scopus	Sí
SpringerLink	No (tras búsqueda preliminar)
Web of Science	Sí

Tabla 4. Selección de la base de datos

Ejecución de la búsqueda

A continuación, se explica el proceso de búsqueda de las publicaciones en las bases de datos seleccionadas.

Para realizar la búsqueda de manera sistemática, definimos criterios de inclusión y criterios de exclusión. Los criterios de inclusión nos permitieron seleccionar inicialmente qué publicaciones de las bases de datos considerar para el estudio, mientras que los criterios de exclusión nos ayudaron a determinar cuáles de esas publicaciones iniciales conservar y cuáles descartar a medida que avanzaba el mapeo de literatura.

Los criterios de inclusión (CI) para este estudio se presentan en la Tabla 5. CI1 se consigue utilizando la cadena de búsqueda. Los CI2, CI3 y CI4 se obtienen aplicando filtros personalizados en cada base de datos.

ID	Criterios de inclusión
CI1	Publicaciones relacionadas con el uso de laboratorios virtuales de programación y otras herramientas en la enseñanza introductoria de programación
CI2	Artículos de conferencias, capítulos de libros y revistas
CI3	Publicaciones entre 2012 y el 22 de febrero de 2023
CI4	Publicaciones disponibles en inglés

Tabla 5. Criterios de inclusión

Los criterios de exclusión (CE) para este estudio se presentan en la Tabla 6. La aplicación de los criterios de exclusión CE1, CE2 y CE3 se realizó utilizando los metadatos proporcionados por los motores de búsqueda, leyendo títulos y resúmenes, y probando enlaces de descarga. En contraste, la aplicación de los criterios de exclusión CE4, CE5, CE6 y CE7 requirió una lectura completa o parcial de las publicaciones para su correcta aplicación.

ID	Criterios de exclusión
CE1	Publicaciones anteriores a 2022 y con menos de dos citas
CE2	Publicaciones no relacionadas con el tema leyendo el título y el resumen
CE3	Publicaciones sin texto completo disponible para leer en línea
CE4	Publicaciones que mencionan cursos centrados en la enseñanza de conceptos considerados demasiado avanzados y no aplicables a un curso de introducción a la programación
CE5	Publicaciones centradas en la enseñanza a estudiantes de niveles inferiores al universitario
CE6	Publicaciones centradas exclusivamente en el desarrollo de herramientas e infraestructuras
CE7	Publicaciones centradas exclusivamente en los contenidos y metodologías de los cursos

Tabla 6. Criterios de exclusión

Con los criterios de inclusión y la cadena de búsqueda bien definidos, realizamos la búsqueda en cada una de las bases de datos. La Tabla 7 presenta la cadena de búsqueda utilizada para cada base

de datos incluida en esta investigación. En el motor de búsqueda directamente solo se pudieron aplicar CI1 y CI3, mientras que CI2 y CI4 tuvieron que ser validados manualmente.

Algunas bases de datos permitían filtros adicionales a las consultas utilizando la cadena de búsqueda. Por ejemplo, Scopus permite incluir el año de publicación y el idioma. Sin embargo, después de consultar la cadena de búsqueda, la mayoría de las bases de datos exigían que esos filtros se aplicaran mediante una interfaz gráfica de usuario. El uso de esa interfaz se describe en la columna de filtros adicionales.

Base de datos	**Cadena de búsqueda**	**Filtros adicionales**
ACM	AllField:(“programming course”) AND AllField:(laboratory) AND AllField:(tool) AND AllField:(university “higher education”)	2012 ≤ año ≤ 2023
IEEE	((“All Metadata”:programming course) AND (“All Metadata”:laboratory) AND ((“All Metadata”:university) OR (“All Metadata”:higher education)) AND (“All Metadata”:tool))	2012 ≤ año ≤ 2023
ScienceDirect	“programming course” AND “laboratory” AND (“university” OR “higher education”) AND “tool”	2012 ≤ año ≤ 2023
Scopus	ALL (“programming course”) AND ALL (laboratory) AND (ALL (university) OR ALL (“higher education”)) AND ALL (tool) AND PUBYEAR > 2011 AND LANGUAGE (english)	
Web of Science	ALL=(“programming course”) AND ALL=(laboratory) AND (ALL=(university) OR ALL=(“higher education”)) AND ALL=(tool) AND LA=(English)	2012 ≤ año ≤ 2023

Tabla 7. Cadenas de búsqueda utilizadas en cada una de las bases de datos seleccionadas

4.1.3 Selección de las publicaciones

El proceso de selección de publicaciones se ejecutó en cinco etapas, presentadas en la Figura 3. Cada etapa representa las publicaciones consideradas para el SMS después de la aplicación de ciertos filtros, y la etapa final representa las publicaciones seleccionadas para este estudio.

Comenzamos con 1323 publicaciones en la etapa 1 (E1), resultado de la aplicación de nuestros criterios de inclusión a las búsquedas en bases de datos. Posteriormente, ejecutamos los siguientes filtros en orden:

(i) Eliminación de duplicados: En primer lugar, se eliminaron las publicaciones duplicadas, resultando en 1272 publicaciones (E2).
(ii) Conteo de citaciones: Luego, se aplicó CE1, eliminando las publicaciones de 2021 o anteriores con menos de dos citas; el recuento de citas se extrajo de la base de datos, y en caso de no disponer del número, se recurrió a Google Scholar para obtener el recuento de citas. Tras realizar esto, quedaron 967 publicaciones (E3).

(iii) Pertinencia: Después, se aplicó CE2, leyendo el título y el resumen de las publicaciones y detectando si estaba alineado con el tema investigado; después de esto, seguían presentes 158 publicaciones (E4).
(iv) Disponibilidad: Luego, se aplicó CE3, eliminando todas las publicaciones que no estuvieran disponibles públicamente o que requirieran una compra adicional para ser consultadas; después de esto, quedaron 150 publicaciones (E5).
(v) Relevancia: Finalmente, se aplicaron CE4, CE5, CE6 y CE7 leyendo las publicaciones y descartando aquellas que, tras una lectura detenida, no eran relevantes para el presente estudio.

Nuestra selección final comprendió 89 publicaciones (E6), de las cuales 33 procedían de ACM, 4 de IEEE, 45 de Scopus y 7 de ScienceDirect.

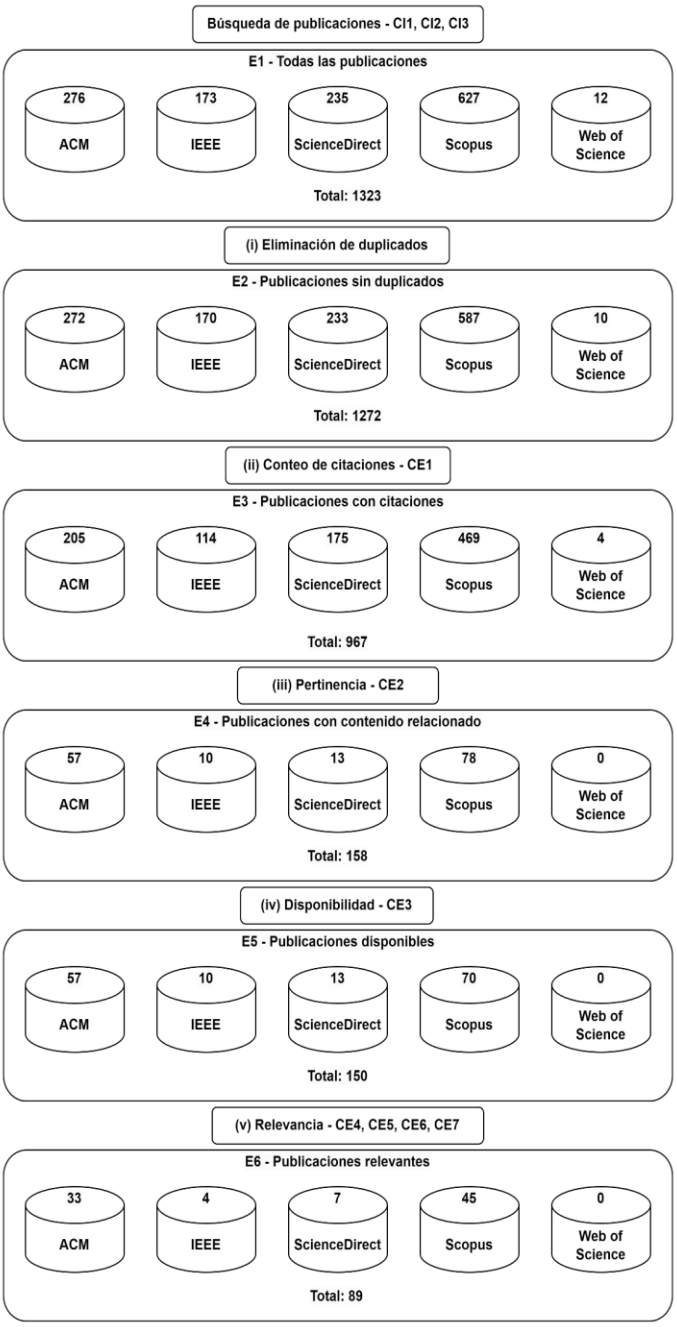

Figura 3. Proceso de selección de las publicaciones

4.1.4 Extracción de datos

Para responder a nuestras preguntas de investigación y aportar información a las investigaciones actuales sobre el tema de interés de este mapeo de literatura, extrajimos la siguiente información de las publicaciones.

Para responder a la PI1, descargamos los metadatos de las publicaciones, como el año de publicación, los autores y el tipo de publicación. Posteriormente, ampliamos esta información con

el número de citas (extraídas de las bases de datos y de Google Scholar) y el país de afiliación del primer autor de la investigación.

Para responder a las PI2, PI3 y PI4, primero realizamos la lectura del título y del abstract de todas las publicaciones, lo cual nos permitió extraer las categorías presentadas en la Tabla 7. Posteriormente, realizamos una lectura completa y minuciosa de cada publicación, lo que nos permitió extraer la información relevante para nuestro estudio y ubicarla en la categoría más afín a la publicación. La Tabla 8 presenta la cantidad de publicaciones que quedaron en cata categoría. Este proceso se realizó con la intención de abordar cada publicación con un enfoque más acorde al contenido presentado.

Categoría	**Descripción**	**Número de publicaciones**
Trabajo relacionado	Mapeos de literatura y revisiones bibliográficas relacionadas con el tema del uso de laboratorios virtuales de programación en la educación. También se incluyeron revisiones de herramientas.	11
Programación visual	Publicaciones relacionadas con el uso de herramientas de programación visual.	2
Visualización de algoritmos	Publicaciones relacionadas con el uso de herramientas de visualización de algoritmos para mejorar la comprensión de los estudiantes.	6
Plagio	Publicaciones centradas en el uso de herramientas para la detección o prevención del plagio.	10
Retroalimentación automática	Publicaciones relacionadas con el uso de herramientas que proporcionan retroalimentación o evaluación automáticas a los estudiantes.	19
Asistencia a estudiantes	Publicaciones en las que se propone o evalúa el uso de una herramienta para ayudar al alumno en su aprendizaje. Estas herramientas pueden adoptar muchas formas, algunas de ellas son IDEs, Chatbots y herramientas de sugerencia.	19
Implicación	Publicaciones relacionadas con el uso de técnicas como la gamificación o la robótica para aumentar el atractivo de los cursos para los estudiantes y su implicación y participación en estos. El término usado en inglés sería *"engagement"*, que corresponde a la capacidad de los cursos de retener la atención y el interés de los estudiantes.	22

Tabla 8. Categorías de las publicaciones de este mapeo de literatura

De las publicaciones extrajimos información general como el año, la institución educativa y el país de la publicación; además de información acerca de qué motivó la investigación, qué herramientas usaron y qué características poseen las herramientas usadas en la publicación. Para ser más asertivos con la información recopilada, también recopilamos algunos datos específicos por categoría tanto para las publicaciones como para herramientas usadas. La Tabla 9 presenta la

información recopilada de las publicaciones y de las categorías, junto con los datos adicionales recopilados según cada categoría tanto para las publicaciones como para las herramientas usadas. La categoría "general" indica que esa información fue recopilada para todas las publicaciones.

Categoría	**Información recopilada de la publicación**	**Información recopilada de las herramientas usadas**
General	▪ Institución ▪ País de la institución ▪ Año de publicación ▪ Nombre del curso ▪ Semestre del curso ▪ Motivación de la publicación ▪ Herramientas usadas y propietario de las herramientas ▪ Características de las herramientas usadas ▪ Herramientas similares mencionadas	▪ Nombre de la herramienta ▪ Publicaciones relacionadas ▪ Disponible para prueba ▪ Información general ▪ Requisito de instalación ▪ ¿Es una app web? ▪ Lenguajes soportados ▪ Gratuita o de pago ▪ Sitio principal de la herramienta ▪ Enlace a la documentación ▪ Materiales disponibles para la enseñanza ▪ Enlace a foro o comunidad ▪ Licencia Open Source ▪ Enlace al código ▪ Información relevante del repositorio ▪ Posibilidad de extender ▪ Dependencia en otras herramientas ▪ Banco de ejercicios preexistente ▪ Capacidad de debugging ▪ Posibilidad de codificar ▪ Posibilidad de ejecutar ▪ Posibilidad de ver el código en ejecución
Programación visual	▪ Metodología de enseñanza usada ▪ Tipo de programación visual (bloques o diagrama de flujo) ▪ Visualización de ejecución	▪ Almacenamiento en nube del programa ▪ Retroalimentación a la hora de programar ▪ Paradigma de programación visual ▪ Características especiales

Visualización de algoritmos	▪ Capacidad para codificar en la herramienta ▪ Características adicionales ▪ Capacidad de debugging de la herramienta ▪ Lenguaje de programación específico	▪ Permite programar ▪ Permite ejecutar ▪ Tipo de visualización ▪ Características adicionales
Plagio	▪ Técnica de detección de plagio utilizada	▪ Algoritmo de detección de plagio ▪ Tiene interfaz ▪ Posibilidad de uso off-line ▪ Características especiales
Retroalimentación automatizada	▪ Tipo de retroalimentación aplicada	▪ Disponibilidad de banco de ejercicios ▪ Posibilidad de codificación ▪ Posibilidad de envío ▪ Tipo de retroalimentación ▪ Tipo de retroalimentación específica
Asistencia a estudiantes	▪ Tipo de asistencia a estudiantes ▪ Características especiales de la herramienta	▪ Tipo de asistencia ▪ Características especiales
Implicación	▪ Estrategia de implicación utilizada ▪ Tipo de implicación utilizada	▪ Características especiales

Tabla 9. Información recopilada acerca de las publicaciones y herramientas

En resumen, para responder a la PI2 se recolectó la información de qué herramientas se están utilizando en las publicaciones que aparecieron en el mapeo de literatura y qué herramientas se mencionan. Para responder a la PI3 se separaron las publicaciones en categorías, para identificar los casos de uso de las herramientas. Para responder a la PI4 realizamos un análisis de las herramientas y de las situaciones encontradas en las distintas publicaciones, que podrían representar retos u oportunidades de mejora.

4.1.5 Síntesis de datos

El proceso de síntesis de datos se centró en analizar y simplificar la información recopilada de las publicaciones presentadas en este SMS para responder a las preguntas de la investigación. Para ello se utilizaron gráficos y herramientas de estadística descriptiva como la media y las frecuencias absolutas.

Para responder a la PI1, analizamos la frecuencia de las publicaciones en distintos países y años y representamos esa información mediante gráficos circulares y de barras.

Para responder a la PI2, se recopilaron datos sobre los lenguajes de programación y herramientas utilizados en los cursos y experimentos documentados en las publicaciones revisadas. Además, se analizó el soporte ofrecido por las herramientas a los diversos lenguajes de programación.

Para responder a la PI3, se categorizó la información de las publicaciones según los diferentes usos de las herramientas de programación en cada fase del proceso educativo. Los resultados se organizaron en tablas y gráficos para una visualización clara y se complementaron con una revisión detallada de cada caso de uso consultando fuentes adicionales.

Para responder a PI4, se identificaron, a partir de los hallazgos obtenidos en las publicaciones revisadas, las principales tendencias y retos en el uso de laboratorios virtuales de programación.

4.2 Informe de resultados

Esta sección presenta los resultados del análisis de la información recopilada sobre las publicaciones seleccionadas para el estudio. Con la información recopilada, pretendemos responder a las preguntas de investigación definidas para el estudio.

En la subsección 4.2.1 presentamos una síntesis de las publicaciones halladas en la categoría "trabajo relacionado" de este mapeo de literatura. En la subsección 4.2.2, respondemos a la pregunta de investigación PI1 sobre los autores, países y frecuencia de publicaciones relacionadas con el uso de laboratorios virtuales de programación para enseñar programación introductoria en entornos universitarios. La subsección 4.2.3 aborda PI2, que examina los lenguajes y herramientas de programación utilizados en cursos introductorios en instituciones universitarias. La subsección 4.2.4 responde a PI3, enfocándose en los fines y fases del proceso educativo en los que se emplean los laboratorios virtuales de programación. Finalmente, en la subsección 4.2.5, se analiza PI4, que explora las tendencias y retos en la implementación de laboratorios virtuales de programación en la educación.

4.2.1 Trabajo relacionado

En esta sección se presenta un análisis de las publicaciones halladas en la categoría "trabajo relacionado" de este mapeo de literatura. Estas publicaciones se analizan de manera independiente y se excluyen de los demás pasos de síntesis y, en general, de las estadísticas generadas en la sección 4.2. Esto se debe a que no aportan datos primarios, sino que se basan en la integración de hallazgos de otros estudios, lo que podría generar redundancia si sus resultados se solapan con los de estudios primarios ya analizados. Además, su enfoque amplio y su propósito de ofrecer una visión general las hacen menos comparables con estudios individuales, lo que podría distorsionar las conclusiones de la síntesis.

Durante la última década, numerosos autores han llevado a cabo estudios de mapeo sistemático (SMS), revisiones sistemáticas de literatura (SLR) y encuestas sobre el uso de metodologías y herramientas para la enseñanza de cursos introductorios de programación en contextos universitarios o similares. Estas investigaciones han abordado múltiples aspectos de este campo, incluyendo la automatización de evaluaciones, herramientas de enseñanza, detección de plagio,

técnicas pedagógicas y visualización de programas, entre otros. Sin embargo, la mayoría de estos trabajos se centra en uno de estos aspectos específicos, lo que limita la obtención de una visión integral. Por ello, las preguntas de investigación planteadas en este SMS son relevantes, ya que los estudios previos no han proporcionado respuestas a algunas de estas. Un resumen de los SMS, SLR y encuestas halladas en la literatura revisada se presenta a continuación.

Automatización de evaluaciones y herramientas

Lajis *et al.* [14] estudiaron la evaluación asistida para la programación práctica, enfocándose en evaluaciones basadas en competencias según la taxonomía de Bloom. Los autores identificaron tres enfoques principales para automatizar la evaluación en materias de programación: el enfoque estático, que analiza el código fuente sin ejecutarlo y evalúa aspectos como el estilo de programación, errores de sintaxis y semántica, detección de palabras clave y detección de plagio; el enfoque dinámico, que evalúa el programa en ejecución considerando errores de programación, diseño del programa y métricas de software; y el enfoque híbrido, que combina ambos métodos. Además, identificaron varias herramientas relacionadas con estos enfoques que ayudan a reducir la carga de evaluación manual para los docentes. Sin embargo, observaron que la mayoría de las evaluaciones automáticas no cuentan con un modelo común de calificación alineado con la taxonomía de aprendizaje.

Ullah *et al*. [15] proporcionaron una visión general de algunas de las herramientas de evaluación automática (AA) existentes en programación, basándose también en los enfoques dinámico, estático e híbrido. Analizaron y compararon diversas herramientas de AA, recopilando información como el tipo de enfoque, los lenguajes soportados, los métodos utilizados, así como las ventajas y limitaciones de cada herramienta. Su estudio destacó que el uso de estos sistemas incrementa significativamente la motivación de los estudiantes para completar las tareas de manera correcta, gracias a la posibilidad de autoevaluarse y recibir retroalimentación inmediata y detallada. Además, estos sistemas disminuyen la carga de trabajo tedioso para los instructores, permitiéndoles dedicar más tiempo a guiar a los estudiantes. Finalmente, sugirieron recomendaciones potenciales sobre las especificaciones necesarias para mejorar las herramientas de evaluación automática en el contexto de la programación para principiantes.

Keuning *et al.* [16] realizaron una revisión sistemática de la literatura (SLR) enfocada en los tipos de retroalimentación proporcionados por herramientas de programación. Clasificaron las herramientas y los mensajes de retroalimentación, reportando los resultados de 101 herramientas. El estudio analizó cómo estas herramientas brindan apoyo en diferentes aspectos, como problemas de estilo, problemas de rendimiento, errores de solución, errores del compilador y el éxito o fracaso en la ejecución de pruebas en los programas de los estudiantes. Concluyeron que la retroalimentación se enfoca principalmente en identificar errores, pero ofrece poca orientación para resolverlos o avanzar en los pasos siguientes. También identificaron que estas herramientas no son fácilmente adaptables a las necesidades específicas de los docentes, lo que limita su utilidad en algunos contextos.

Plagio y detección de similitudes

Albluwi [17] presentó una revisión sistemática sobre el plagio en la educación informática, identificando una gran cantidad de artículos que abordan tanto estrategias para reducir las oportunidades de plagio como herramientas diseñadas para detectarlo. Los artículos analizados se categorizaron en función de conceptos relacionados con el triángulo del fraude: presión,

oportunidad, racionalización y otras categorías adicionales. Este modelo proporciona una base teórica para entender cuándo es más probable que ocurra el fraude académico.

Por su parte, Matija *et al.* [18] realizaron una revisión sistemática enfocada en las herramientas utilizadas para detectar similitudes en código fuente dentro del ámbito académico. Su análisis se centró en identificar los métodos de ofuscación que los estudiantes emplean al plagiar, las herramientas y algoritmos utilizados para detectar casos de plagio, los conjuntos de datos y métricas aplicados para evaluar estas herramientas, así como los lenguajes de programación principales en los que estas se enfocan. El estudio identificó 16 métodos generales de ofuscación y destacó las cinco herramientas más utilizadas en el ámbito académico: JPlag, Marble, MOSS, Plaggie y SIM. El artículo concluye que se necesita brindar un mayor soporte a lenguajes distintos de Java y C/C++, ya que estos dominan en las herramientas de detección de similitudes académicas. Asimismo, los autores sugieren cambiar el término "detección de plagio" por "detección de similitudes" para evitar malentendidos y estigmatizaciones. Finalmente, enfatizan la importancia de priorizar la prevención del plagio por encima de su detección, proponiendo la implementación de entregas ágiles mediante sistemas de control de versiones, como Git y SVN, para rastrear el progreso y la autoría de los entregables.

Enseñanza de programación introductoria

Luxton-Reilly *et al.* [19] realizaron una revisión sistemática de la literatura sobre programación introductoria, analizando 1666 artículos. Su enfoque principal fue examinar las habilidades de los estudiantes y sus percepciones hacia los cursos introductorios de programación, las herramientas, técnicas y estructuras de los cursos utilizadas para su enseñanza, las razones y métodos para seleccionar un lenguaje o paradigma de programación, y los diferentes métodos de evaluación y herramientas utilizadas. Para ello, clasificaron los artículos en las categorías de "estudiante", "enseñanza", "currículo" y "evaluación", permitiendo que un artículo pudiera pertenecer a varias categorías. Los autores destacaron que algunos campos de investigación no muestran un crecimiento significativo, probablemente debido a la falta de innovación o controversia en esos temas. Por el contrario, áreas como el desarrollo de herramientas están muy activas. Sin embargo, señalaron que, aunque el desarrollo de herramientas es popular, hay poca investigación enfocada en su evaluación. Además, encontraron un contraste entre la satisfacción reportada en estudios relacionados con los estudiantes y los niveles de compromiso observados, junto con altas tasas de deserción.

Idongesit Etenga *et al.* [20] llevaron a cabo una revisión para identificar las prácticas y herramientas empleadas en la enseñanza de programación en contextos con recursos limitados. Encontraron que algunos cursos no incluyen un lenguaje de programación en el primer semestre, y que Java, Python y C/C++ son los lenguajes más populares en los primeros semestres de educación en programación. Estos lenguajes suelen combinarse con metodologías de enseñanza como el aprendizaje colaborativo, el aprendizaje combinado *(blended learning)* y la programación en parejas. Además, realizaron recomendaciones sobre qué lenguajes de programación enseñar y qué objetivos establecer según el semestre en el que se encuentren los estudiantes.

Araujo *et al.* [21] revisaron 27 artículos relacionados con el desarrollo y la evaluación de habilidades de pensamiento computacional en la educación K-12. Este análisis abordó las herramientas utilizadas, los contenidos enseñados y los temas evaluados en los cursos. Encontraron que las preguntas de opción múltiple son el método de evaluación más utilizado, seguidas por las

tareas de codificación, y destacaron que el desarrollo de habilidades para resolver problemas es el objetivo principal de la educación en programación K-12. También señalaron que herramientas como Scratch, App Inventor y Alice son populares debido a que eliminan la preocupación por la sintaxis, pero advirtieron que estas herramientas están limitadas en su enfoque pedagógico, restringiéndose principalmente a la resolución de problemas básicos, narración digital y diseño de juegos.

Visualización de programas

Sorva *et al.* [22] llevaron a cabo un análisis de los sistemas de visualización de programas desarrollados entre 1983 y 2013, diseñados para enseñar a principiantes sobre el comportamiento de los programas durante su ejecución. Identificaron 24 herramientas, entre ellas jGrasp, PythonTutor, CMeRun y PlanAni, recopilando datos relevantes como fecha de creación, estado de actividad, propósito, paradigma, lenguaje compatible, sistema de evaluación estudiantil y metodología de compromiso. Para evaluar estas metodologías, crearon una taxonomía de compromiso en dos dimensiones (2DET) que clasifica las herramientas según su "compromiso directo" y "compromiso de propiedad". Además, algunas herramientas fueron revisadas en profundidad, acompañadas de capturas de pantalla como ejemplos ilustrativos.

Las conclusiones del estudio destacan que, en general, los sistemas de visualización para principiantes suelen ser prototipos de corta duración, enfocados en ofrecer una visualización controlada por el usuario de las animaciones de los programas. Aunque los resultados de las evaluaciones apoyan ampliamente el uso de estos sistemas en la educación de programación introductoria, la investigación existente no permite extraer conclusiones detalladas sobre su impacto en el compromiso de los estudiantes. Además, la mayoría de las herramientas están diseñadas para ser compatibles con un único lenguaje, lo que limita su aplicabilidad. También señalaron la necesidad de investigar más sobre las interacciones entre las herramientas de visualización, los estudiantes, los entornos de aprendizaje, las formas de compromiso y los objetivos de aprendizaje específicos.

El estudio identificó que las herramientas más populares son aquellas similares a los depuradores (debuggers). Finalmente, los autores propusieron que desarrollar una nueva herramienta que integre las mejores ideas de los prototipos anteriores representaría una contribución significativa al campo.

Estudios en áreas específicas

Lukkarinen *et al.* [23] llevaron a cabo una revisión sobre la enseñanza de la programación orientada a eventos (EDP, por sus siglas en inglés) enfocada en principiantes. Analizaron un total de 105 artículos que cubren diversos aspectos relacionados con EDP, como prácticas de enseñanza, recursos de aprendizaje, herramientas de software o bibliotecas para apoyar la enseñanza, y estudios empíricos. El artículo señala que, aunque la mayoría de los estudios se centran en el nivel universitario, también existe una cantidad considerable de trabajos enfocados en la educación K-12. Sin embargo, destaca que es poco común que EDP sea el tema principal de un curso; más frecuentemente, se aborda como un tema secundario dentro de los cursos introductorios de informática, como CS1 o CS2.

En su análisis, los autores identificaron 37 lenguajes de programación utilizados para enseñar EDP, siendo los más populares Java, App Inventor, C++ y Scratch. Asimismo, destacaron tres pilares

principales en esta área: las interfaces gráficas de usuario (GUI), los gráficos computacionales y el desarrollo de robótica. Finalmente, concluyen que la enseñanza de EDP es un área con limitada investigación empírica. Por ello, proponen al final de su artículo algunas ideas para futuras investigaciones que podrían contribuir al desarrollo y comprensión de este campo.

4.2.2 Análisis demográfico

Esta sección pretende responder a la pregunta de investigación PI1, que indaga sobre los autores, los países y la frecuencia de las publicaciones presentes en este estudio. También se presenta el análisis correspondiente a los elementos demográficos de las publicaciones seleccionadas, como (i) países de origen de las publicaciones, (ii) autores, (iii) distribución temporal de las publicaciones, y (iv) publicaciones más citadas.

Países

Según los resultados del estudio, los 5 países que más publicaciones tienen sobre el uso de herramientas para la enseñanza de la programación introductoria en la educación superior son (i) Estados Unidos (19 publicaciones, 21%), (ii) España (8 publicaciones, 9%), (iii) Nueva Zelanda (7 publicaciones, 7%), (iv) Finlandia (3 publicaciones, 3%) y (v) Suráfrica (3 publicaciones, 3%). Estos primeros 5 países suman un 44% de las publicaciones analizadas. Entre los demás países con publicaciones se encuentran Serbia, Malasia, Moroco, Italia, Australia, Arabia Saudita, Rusia, Alemania, China, Indonesia, etc. En total hay 40 países con publicaciones presentes en este estudio.

El análisis se realizó utilizando el país de afiliación del primer autor de cada publicación seleccionada. Estos resultados se resumen en la Figura *4*, donde cada porción del diagrama corresponde a la proporción de publicaciones originadas en cada país.

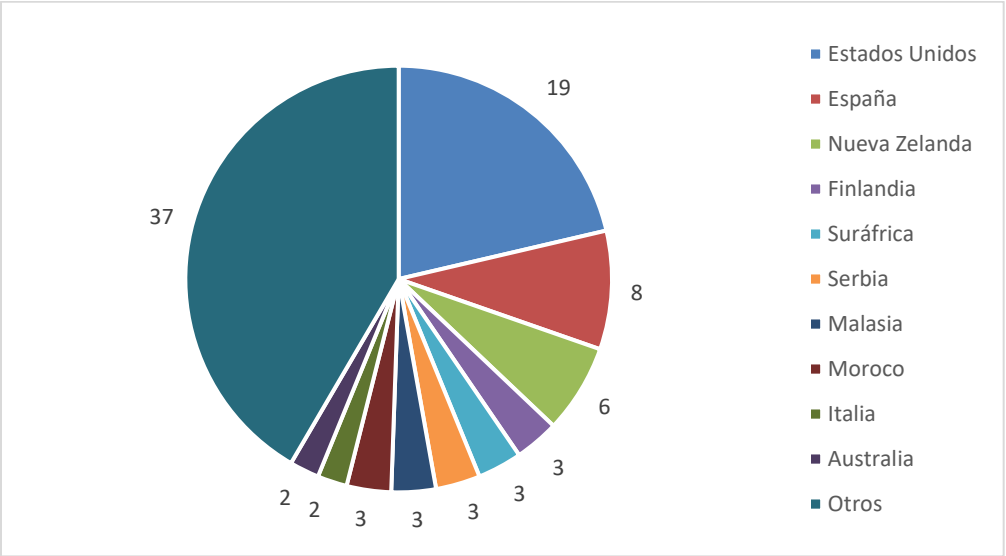

Figura 4. Número de publicaciones por país

Autores

Para realizar el análisis de autores, se empleó código en Python utilizando las bibliotecas pandas y networkx. Los datos fueron descargados en formato CSV para facilitar su procesamiento, y se analizaron únicamente los documentos incluidos en esta revisión de literatura.

En el análisis de las relaciones entre autores dentro del conjunto de datos seleccionado, emergen varias figuras influyentes y patrones de colaboración. Simon [19] y Andrew Luxton-Reilly [19], [24] se encuentran entre los autores mejor conectados según su centralidad de grado, lo que significa que colaboran con frecuencia con una variedad de otros investigadores.

Cabe destacar que Simon, junto con Oscar Karnalim [25], [26], [27], [28] y Paul Denny [24], [29] también tiene una posición alta en centralidad de intermediación, lo que los coloca como un puente entre varios clústeres de autores. Estos autores contribuyen a una red de investigación cohesiva, lo que indica que probablemente son fundamentales para conectar y facilitar el intercambio de conocimientos entre grupos de investigación.

Las parejas comunes de coautores, como Thomas W. Price y Samiha Marwan [30], [31], [32], quienes colaboraron en tres artículos, así como José Paulo Leal y Ricardo Queirós [33], [34], quienes trabajaron juntos en dos artículos, indican sólidas asociaciones académicas en el campo. Estas colaboraciones repetidas sugieren esfuerzos de investigación enfocados y potenciales contribuciones conjuntas de impacto.

Distribución temporal

La Figura 5 muestra la distribución temporal y el tipo de publicaciones presentes en el estudio. Esta figura muestra un aumento del número de publicaciones en los años 2016, 2018, 2020 y 2022. El año 2022 presenta el mayor número de publicaciones, con 15 publicaciones, seguido del año 2018 con 13 publicaciones y del año 2016 con 11 publicaciones.

En el 2020, hubo un aumento de las publicaciones, probablemente relacionada con la pandemia de coronavirus. En el año de 2022 se presenta un pico de publicaciones que puede atribuirse a que a este año no le aplicamos CE1, debido a que las publicaciones llevaban poco tiempo publicadas y consideramos que no era suficiente para que fueran citadas aún. En el año 2023 hay pocas publicaciones debido a que estas se extrajeron durante el primer trimestre de 2023.

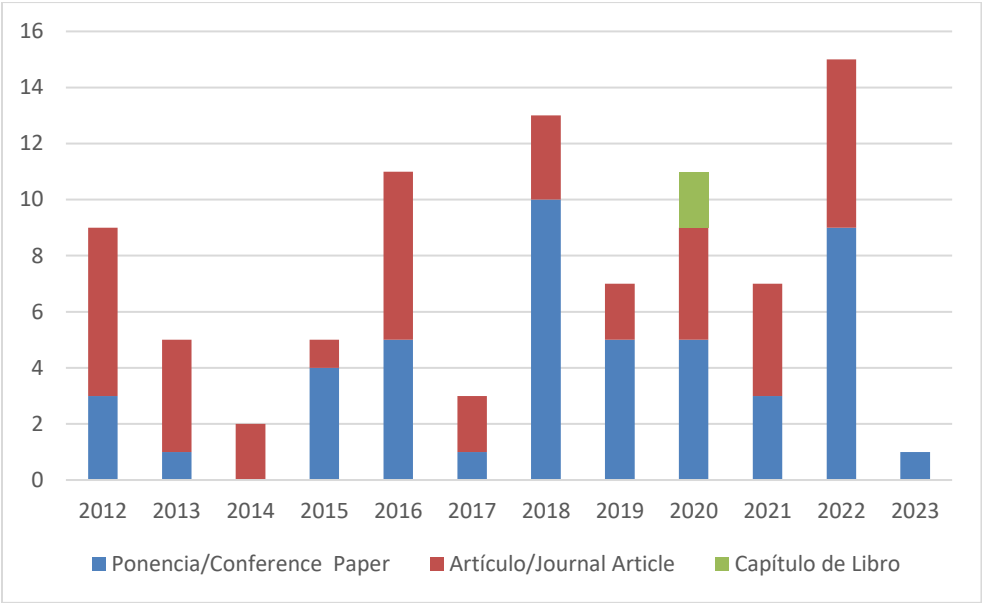

Figura 5. Distribución temporal de las publicaciones

Publicaciones más citadas

La Tabla 10 muestra las diez publicaciones más citadas, que aparecen en el estudio, con su respectivo año de publicación, número de citaciones y categoría de publicación. El número de citas se basa en la información encontrada en sus respectivas bases de datos hasta marzo de 2023. Para aquellas publicaciones que no contaban con esa información en su respectiva base de datos, se extrajo la información de Google Scholar.

Las tres publicaciones más citadas son: (i) "Improving programming skills in engineering education through problem-based game projects with Scratch", con 115 citas, (ii) "Protus 2.0. Ontology-based semantic recommendation in programming tutoring system", con 103 citas y "A distributed system for learning programming on-line", con 96 citas.

Los resultados muestran que 3 de las 10 primeras publicaciones pertenecen a la categoría de “retroalimentación automática”, 2 a la de “programación visual”, 2 a la de “asistencia a estudiantes”, 2 a la de “implicación” y 1 a la de “visualización de algoritmos”. Ninguna de las 10 primeras publicaciones pertenece a la categoría de “plagio”.

Ref.	**Título**	**Año de publicación**	**Citaciones**	**Categoría**
[35]	Improving programming skills in engineering education through problem-based game projects with Scratch	2018	115	Programación visual
[36]	Protus 2.0: Ontology-based semantic recommendation in programming tutoring system	2012	103	Asistencia a estudiantes

[34]	A distributed system for learning programming on-line	2012	96	Retroalimentación automática
[37]	Mediated Transfer: Alice 3 to Java	2012	69	Programación visual
[38]	PlayIT: Game based learning approach for teaching programming concepts	2016	62	Implicación
[39]	Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1	2017	56	Visualización de algoritmos
[40]	Coderunner: A Tool for Assessing Computer Programming Skills	2016	46	Retroalimentación automática
[41]	Measuring Code Behavioral Similarity for Programming and Software Engineering Education	2016	40	Retroalimentación automática
[42]	An affective and Web 3.0-based learning environment for a programming language	2018	39	Asistencia a estudiantes
[32]	Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science	2020	29	Implicación

Tabla 10. Publicaciones más citadas

4.2.3 Lenguajes y herramientas

Para responder a la pregunta sobre qué lenguajes y herramientas se utilizan en los cursos introductorios de programación (PI2), recopilamos datos sobre los lenguajes de programación y herramientas usados en los cursos y experimentos de las publicaciones. También analizamos qué lenguajes son compatibles con estas herramientas. Los resultados de nuestra investigación se presentan a continuación.

Lenguajes usados

Para el análisis de los lenguajes utilizados en los experimentos de las publicaciones, se extrajo información sobre el lenguaje de programación empleado a través de la lectura detallada de cada publicación. En casos donde no se mencionaba explícitamente el lenguaje, se pudieron inferir a partir de ejemplos de los ejercicios resueltos por los estudiantes. La Figura 6 muestra el número de publicaciones por lenguaje y por año.

Los lenguajes más populares en los estudios son Java (18 publicaciones), Python (13 publicaciones), C (12 publicaciones) y C++ (12 publicaciones). Es importante destacar que, aunque C y C++ comparten similitudes, se consideran lenguajes distintos en este estudio debido a sus enfoques educativos. Con C, generalmente se enseña algoritmia imperativa básica, mientras que C++ se utiliza para profundizar en la programación orientada a objetos.

Además, se identificaron lenguajes gráficos como Scratch (4 publicaciones) e iSnap (3 publicaciones), una evolución de Scratch.

Cabe señalar que desde 2018, Python ha ganado relevancia, con un notable incremento en su uso en 2022, probablemente debido a su practicidad y creciente importancia en el ámbito educativo e investigativo.

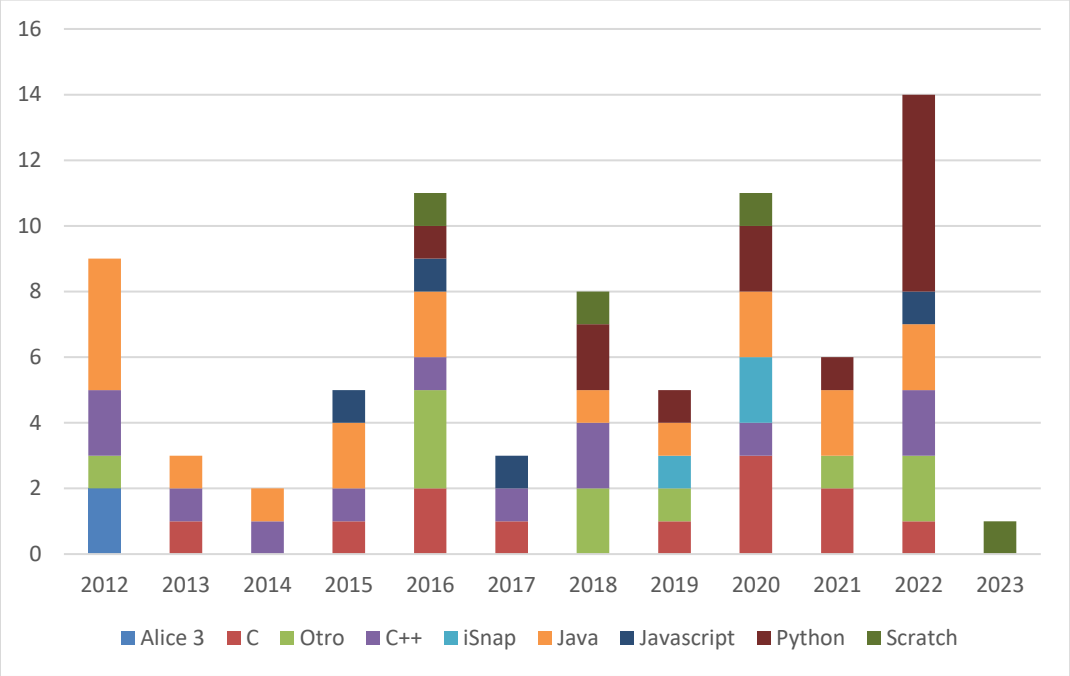

Figura 6. Evolución temporal de los distintos lenguajes de programación por año

La Figura 7 muestra los paradigmas de programación enseñados en los distintos cursos analizados. El paradigma más representado es la programación orientada a objetos (POO), con 45 publicaciones, principalmente utilizando lenguajes como Java, Python, C# y C++. A continuación, se encuentra la programación imperativa, con 17 publicaciones, que utiliza lenguajes como C, Javascript y Pascal. Un tercer grupo incluye herramientas visuales, con 12 publicaciones, en las que se emplean lenguajes como Scratch, iSnap (una alternativa a Scratch), Alice y otros lenguajes gráficos propietarios. Cabe señalar que, aunque el "paradigma de programación visual" no existe formalmente, se incluye aquí para diferenciar claramente el uso de lenguajes gráficos de los demás lenguajes utilizados, a pesar de que estos lenguajes pueden tener fundamentos en programación imperativa. Finalmente, la programación funcional, representada solo por Haskell y Lisp, tiene una presencia menor, con solo 3 publicaciones en este estudio.

En general, la Figura 7 muestra que en las publicaciones seleccionadas existe un gran interés por los paradigmas de programación imperativa y orientada a objetos, ocasionalmente facilitados por

el uso de lenguajes de programación visuales. En contraste, se observa un menor interés por los lenguajes de programación funcionales, con una representación significativamente más baja en las publicaciones analizadas.

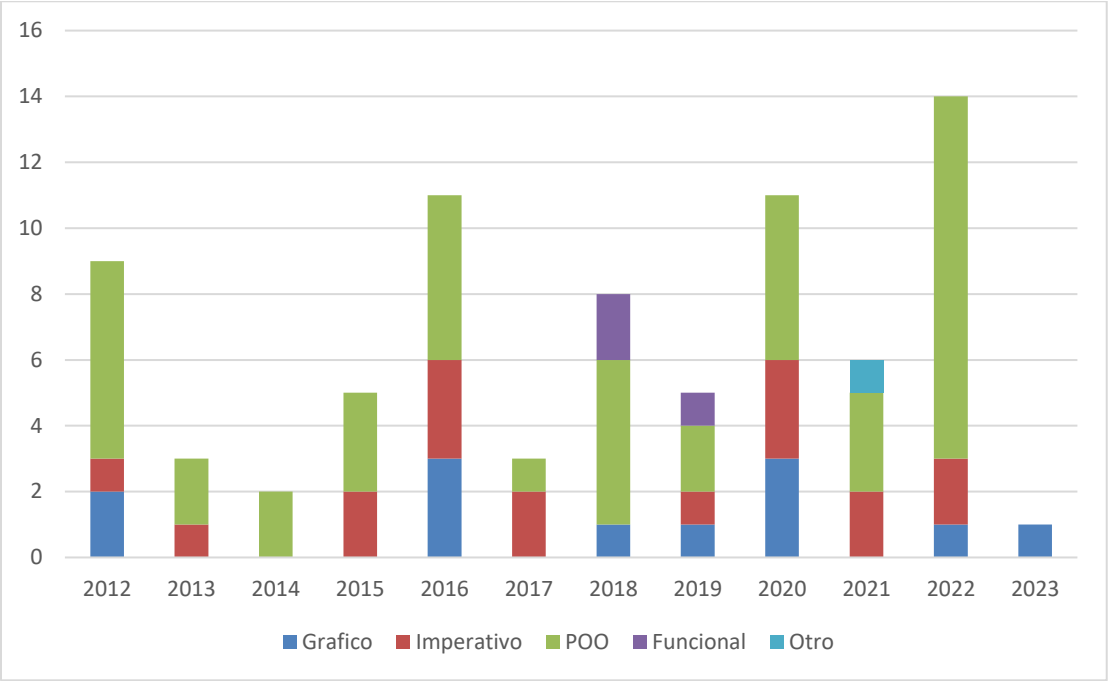

Figura 7. Evolución temporal de los distintos paradigmas de programación en la educación

Herramientas usadas

Esta sección presenta un análisis de las herramientas de programación halladas en las publicaciones de este estudio. Se incluyen información sobre características principales, como los lenguajes de programación que soportan, su disponibilidad como software de código abierto, su capacidad de extensibilidad y el nivel de mantenimiento recibido.

Para recolectar la información sobre las herramientas, se siguió un proceso sistemático que se ilustra en la
Figura 8. Primero, se verificó si el documento incluía enlaces directos para consultar, probar o acceder al código de la herramienta. En caso de que la herramienta proviniera de otra publicación, se repetía este mismo proceso en las referencias correspondientes. Si no se encontraba un enlace directo, se realizaba una búsqueda exhaustiva en internet utilizando el nombre de la herramienta. Para cada herramienta localizada, se invirtió un máximo de dos horas en su prueba y, si era necesario, en la instalación correspondiente. Generalmente, las herramientas sin nombre en las publicaciones no estaban disponibles para el público, mientras que aquellas con nombre indicaban una intención de distribución pública o mantenimiento activo. La Figura 8 representa este proceso por medio de un diagrama de flujo.

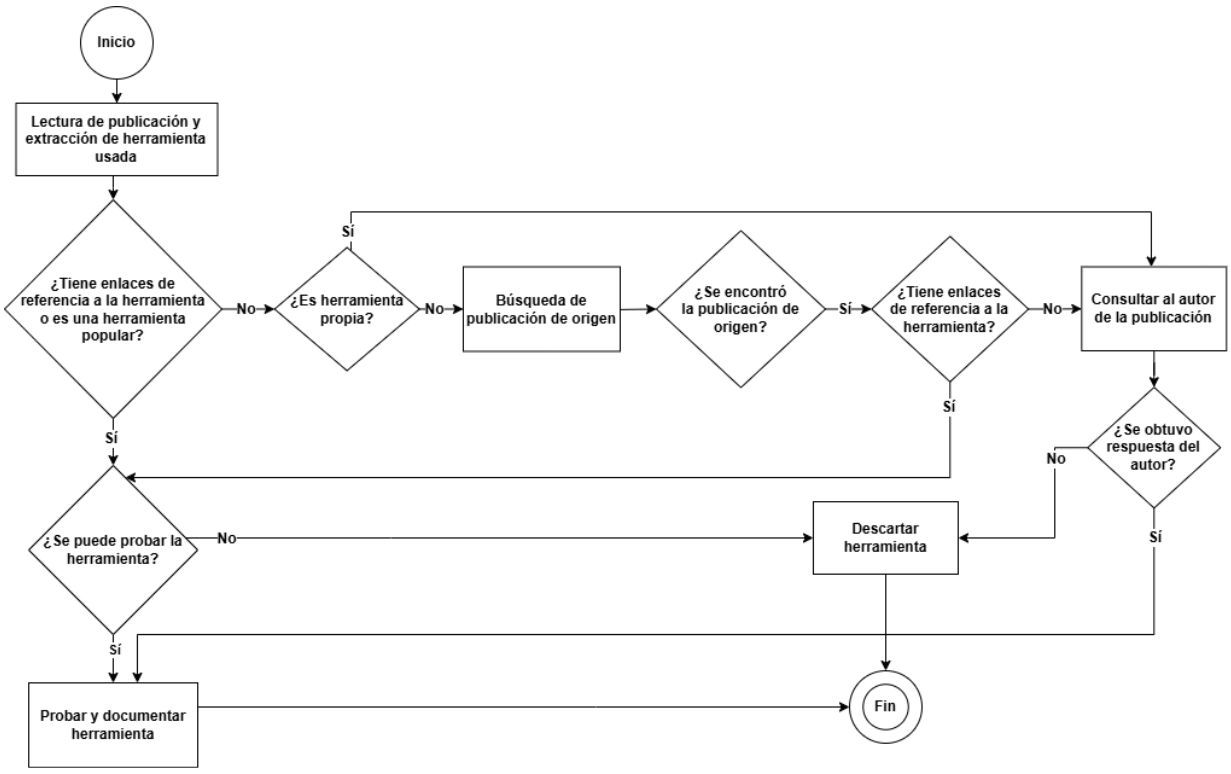

Figura 8. Proceso de mapeo de herramientas

La Tabla 11 presenta las publicaciones con herramientas que no se pudieron probar. En total, se descartaron 47 herramientas por diversas razones. De estas, 32 no estaban disponibles para uso público: en algunos casos, solo se halló la documentación sin acceso al código, resultaron inaccesibles debido a requisitos específicos de hardware o software, o bien se hallaba prohibido su acceso al público general. 11 herramientas adicionales no se encontraron en internet o no estaban en un lenguaje compatible con esta investigación. Finalmente, 4 herramientas no se pudieron probar debido a que dependen de servidores o servicios que ya no encuentran operativos.

Motivo de descarte	Referencias
No disponible	[24], [27], [29], [30], [31], [32], [38], [39], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63]
No hallada	[36], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73]
No operativa	[34], [41], [74], [75]

Tabla 11. Motivo de descarte de herramientas

En la Tabla 12 se presentan las herramientas que cuentan con 2 o más usos en las publicaciones de este estudio, acompañadas de una breve descripción de sus características más relevantes.

Las herramientas más destacadas incluyen Scratch, con 4 usos; MOSS e iSnap, con 3 usos cada una; y Alice, JUnit, PythonTutor, Lightbot y SnatchBot, cada una con 2 usos. Entre estas herramientas, 3 son de programación visual (Scratch, iSnap y Alice), 1 es para la detección de plagio (MOSS), 1 para la visualización de algoritmos (PythonTutor), 1 es una biblioteca de pruebas unitarias (JUnit), otra gamifica la programación (Lightbot) y, finalmente, SnatchBot es una aplicación pública para la creación de chatbots. Entre estas herramientas, 6 son gratuitas, cuentan con buen soporte y una documentación adecuada, lo que resalta la relevancia de estos aspectos en los entornos educativos.

Nombre	**Descripción**	**Características**	**Publicaciones**
Scratch	Scratch es un lenguaje de programación visual de alto nivel basado en bloques y una plataforma web dirigida principalmente a niños como herramienta educativa para aprender programación, con un público objetivo entre los 8 y 16 años. Página principal: https://scratch.mit.edu/	Gratuito Pataformas: Web, nativo Buen soporte Buena documentación De código abierto Soporta programación visual	4 publicaciones: [35], [47], [69], [74]
MOSS	MOSS (Measure Of Software Similarity) es un sistema automático para determinar la similitud de programas. Hasta la fecha, la principal aplicación de MOSS ha sido la detección de plagio en clases de programación. Desde su desarrollo en 1994, MOSS ha demostrado ser muy efectivo en este rol. El algoritmo detrás de MOSS representa una mejora significativa respecto a otros algoritmos de detección de trampas. Página principal: https://theory.stanford.edu/~aiken/moss/	Gratuito Pataformas: Web Buen soporte Buena documentación Lenguajes soportados: C, C++, Java, C#, Python, VisualBasic, Javascript, Haskell, etc.	3 publicaciones: [76], [77], [78]

iSnap	Snap! es un lenguaje de programación ampliamente accesible tanto para niños como adultos, que también funciona como una plataforma para el estudio serio de la informática. Página principal: https://snap.berkeley.edu/	Gratuito Pataformas: Web Buen soporte Buena documentación De código abierto Soporta programación visual	3 publicaciones: [30], [31], [32]
Alice	Alice es un lenguaje de programación educativo basado en objetos, con un entorno de desarrollo integrado (IDE). Alice utiliza un entorno de arrastrar y soltar para crear animaciones por computadora usando modelos en 3D. El software fue desarrollado inicialmente en la Universidad de Virginia en 1994 y, posteriormente, en la Universidad Carnegie Mellon (desde 1997), por un grupo de investigación liderado por Randy Pausch (de Wikipedia). Página principal: http://www.alice.org/	Gratuito Pataformas: Nativo Buen soporte Buena documentación De código abierto Soporta programación visual	2 publicaciones: [37], [79]
JUnit	JUnit es un marco de pruebas unitarias para el lenguaje de programación Java. JUnit ha sido fundamental en el desarrollo de la metodología de desarrollo guiado por pruebas y es parte de una familia de marcos de pruebas unitarias conocida como xUnit, que se originó con SUnit. Página principal: https://junit.org/junit5/	Gratuito Pataformas: Nativo Buen soporte Buena documentación De código abierto Lenguajes soportados: Java	2 publicaciones: [72], [80]

PythonTutor	PythonTutor es una herramienta de visualización de programas para múltiples lenguajes de programación. Página principal: https://pythontutor.com/	Gratuito Pataformas: Web Buen soporte Buena documentación De código abierto Lenguajes soportados: Python, JavaScript, C, C++ y Java	2 publicaciones: [28], [81]
Lightbot	LightBot es un juego de rompecabezas basado en la codificación; enseña de manera oculta la lógica de programación mientras el jugador avanza en el juego. Página principal: https://lightbot.com/	De pago Pataformas: Móvil Soporta programación visual	2 publicaciones: [38], [82]
SnatchBot	SnatchBot es una plataforma educativa para crear chatbots, ideal para enseñar conceptos de programación y procesamiento de lenguaje natural. Con una interfaz accesible, permite a los estudiantes aprender lógica y diseño de interacciones conversacionales, aplicando conocimientos en proyectos reales de inteligencia artificial sin necesidad de experiencia avanzada en programación. Página principal: https://snatchbot.me/	Gratuito Pataformas: Web	2 publicaciones: [71], [75]

Tabla 12. Herramientas más relevantes de acuerdo a su uso

Entre las aplicaciones adicionales se encuentran herramientas con distintos propósitos educativos. Algunas de estas son entornos de desarrollo integrados (IDE), como BlueJ, Greenfoot y Replit, que se utilizan para escribir y ejecutar código en entornos educativos, permitiendo a los estudiantes experimentar con la programación en un entorno controlado. Otras aplicaciones funcionan como jueces automáticos para la evaluación de código, como CodeRunner, Jutge.org, Sharif Judge y Mooshak, que no solo verifican la precisión de los programas mediante pruebas automáticas, sino

que también soportan una gran variedad de lenguajes de programación, incluyendo algunos funcionales, lo que aumenta su versatilidad en cursos de introducción a diversos paradigmas.

Asimismo, hay librerías de pruebas y automatización, como JUnit, QuickCheck, XUnit y Pylint, que se emplean principalmente para automatizar pruebas y facilitar la detección de errores en el código, ayudando a los estudiantes a identificar y corregir fallos de forma eficiente.

En general, se observa que las aplicaciones con funcionalidades de evaluación automática son ampliamente utilizadas en contextos educativos, ya que permiten una retroalimentación rápida y objetiva sobre el desempeño del estudiante. Las herramientas de visualización de ejecución de código y de programación visual también son populares, pues facilitan el aprendizaje al permitir a los estudiantes ver en tiempo real cómo se comporta su código o trabajar en entornos visuales accesibles. Las herramientas de código libre, bien documentadas y con actualizaciones regulares, destacan especialmente por su accesibilidad y adaptabilidad en diversos contextos educativos. No obstante, algunas herramientas propietarias, como Replit, Jupyter y MOSS, sobresalen por la calidad de su documentación y mantenimiento, lo que las convierte en alternativas efectivas a pesar de no ser open-source.

4.2.4 Casos de uso

En esta sección damos respuesta a la PI3 sobre los fines y fases del proceso educativo en que se utilizan los laboratorios virtuales de programación y profundizamos en los diversos casos de uso identificados como categorías en la Tabla 8 de la Sección 4.1.4.

Identificamos un conjunto de situaciones en las que se utilizan herramientas en cursos introductorios de programación, que son: (i) facilitar la programación visual y reducir la barrera de la sintaxis, que suele ser uno de los mayores obstáculos en el aprendizaje de programación; (ii) ofrecer visualización de algoritmos, para comprender mejor la ejecución de un programa, crear esquemas mentales de estructuras de datos y encontrar errores; (iii) identificar y prevenir el plagio; (iv) proporcionar retroalimentación automática, lo cual minimiza la carga laboral del docente y mejora la experiencia de los estudiantes ofreciendo una retroalimentación oportuna; (v) asistir a los estudiantes en la resolución de errores y comprensión de conceptos; y (vi) fomentar la implicación y motivación de los estudiantes para que participen en las actividades y estudien activamente.

I) Programación visual

La programación visual es una estrategia fundamental en cursos introductorios, especialmente para estudiantes que recién comienzan en programación. Este enfoque, que utiliza interfaces gráficas, diagramas de flujo y bloques en lugar de código textual, facilita el aprendizaje al reducir la carga cognitiva relacionada con la sintaxis y los errores de escritura. Así, el proceso se enfoca en la lógica y la estructura de la programación, dejando de lado las complejidades sintácticas iniciales. Esto no solo mejora la comprensión de conceptos básicos, sino que también aumenta la motivación y confianza de los estudiantes al hacer la programación más accesible.

En el estudio [37], el uso combinado de Alice 3 y Java, junto con la teoría de transferencia mediada, permite a los estudiantes aplicar en Java los conocimientos adquiridos en Alice, reforzando su comprensión. Durante dos años, esta pedagogía mostró mejoras significativas en las calificaciones de las secciones experimentales frente a las secciones sin tratamiento. De manera similar, Damla

et al. [35] utilizaron Scratch en proyectos de desarrollo de juegos en cursos introductorios de programación, lo que resultó en una mejora significativa en el rendimiento de los estudiantes al combinar el aprendizaje basado en juegos y problemas reales.

En el ámbito educativo, los lenguajes de programación visual más populares son Scratch, que se visualiza en la Figura 9, (junto con sus variantes, como iSnap) y Alice, ambos basados en la programación por bloques. Existen también lenguajes visuales enfocados en aplicaciones específicas, como Fluxprog [83] y Lego Mindstorms [79], que se utilizan ampliamente en la robótica educativa. Estos lenguajes permiten a los estudiantes programar y controlar robots mediante entornos visuales, integrando conceptos de programación y tecnología en proyectos prácticos y motivadores.

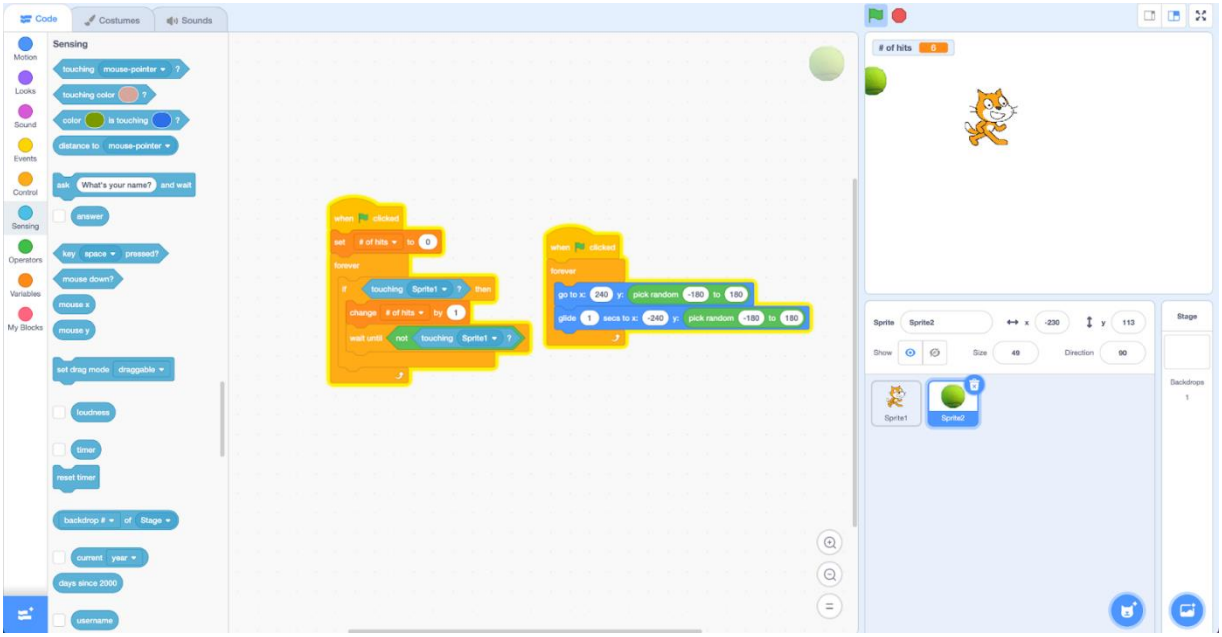

Figura 9. Ejemplo de programación visual usando Scratch

La popularidad de los lenguajes basados en bloques refleja una tendencia en la industria educativa a preferir esta estructura visual sobre los demás tipos de representación visual como son los diagramas de flujo.

En este estudio, 10 publicaciones utilizaron lenguajes de programación visual, lo que evidencia su relevancia en la enseñanza de programación. Aunque solo 2 publicaciones se enfocaron directamente en ellos, el uso de herramientas de programación visual se refleja ampliamente en la literatura. Estas herramientas destacan por su capacidad para facilitar la enseñanza y sus efectos positivos en el rendimiento académico y la motivación de los estudiantes, sugiriendo que su inclusión en cursos introductorios beneficia tanto el aprendizaje inmediato como el desarrollo de habilidades a largo plazo.

II) Visualización de algoritmos

La visualización de algoritmos es una estrategia educativa que utiliza representaciones gráficas o depuradores interactivos para mostrar el flujo de ejecución y comportamiento de los algoritmos, facilitando la comprensión de su lógica y estructura. En los cursos de programación, esta técnica

permite a los estudiantes observar cómo se desarrollan los procesos internos de un programa, desde el manejo de estructuras de datos hasta el control de flujo y la recursividad. Al hacer visibles los pasos de un algoritmo, los estudiantes pueden identificar errores de manera intuitiva y mejorar su comprensión conceptual, lo cual es especialmente útil en etapas introductorias. La visualización de algoritmos no solo refuerza el aprendizaje de conceptos abstractos, sino que también promueve un entendimiento profundo y duradero de los principios de la programación.

Las herramientas de visualización gráfica suelen emplear animaciones para ilustrar conceptos, ayudando a los estudiantes a observar el comportamiento de un programa paso a paso. Estas visualizaciones permiten comprender la evolución de las variables y la estructura del flujo sin necesidad de escribir código. Ejemplos de este tipo de herramientas incluyen Jsvee [84], PlanAni [85] y PLTutor[39]. Estas herramientas muestran procesos complejos a través de animaciones, haciendo más accesible la comprensión de los algoritmos. En particular, PlanAni adopta un enfoque único en la visualización de algoritmos al representar visualmente los roles de las variables. En lugar de simplemente mostrar el valor de las variables, PlanAni (visible en la Figura *10*) utiliza símbolos visuales distintivos: por ejemplo, representa un índice como una serie de pisadas que avanzan, o un valor booleano como una bombilla que se enciende y apaga.

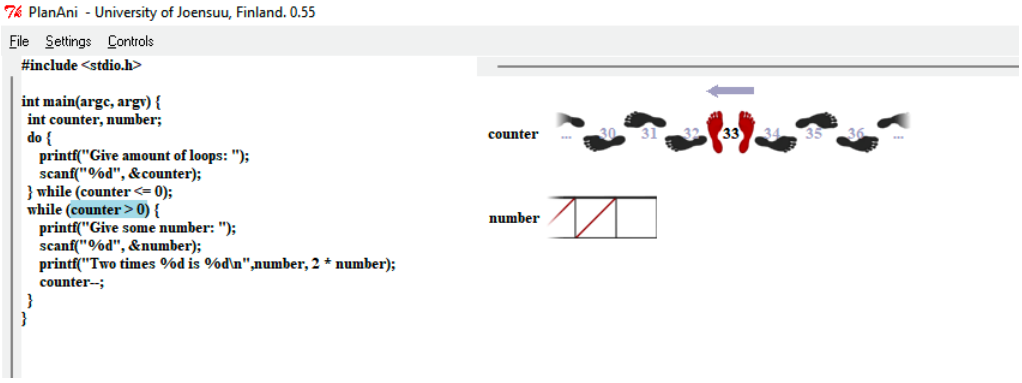

Figura 10. Representación visual de un algoritmo en PlanAni

Por otro lado, los depuradores interactivos proporcionan un enfoque más versátil al permitir que los estudiantes inspeccionen variables, adelanten o retrocedan la ejecución y visualicen estructuras de datos de manera sencilla. Aunque no suelen ofrecer animaciones detalladas, estos depuradores permiten a los estudiantes una manipulación directa del código, facilitando la práctica activa en la detección y corrección de errores. Ejemplos de depuradores incluyen DS-PITON [26], PITON [26] y PythonTutor [81], siendo PythonTutor una de las herramientas más populares a nivel educativo para la demostración de ejecución de código paso por paso en Python, su interfaz se puede apreciar en la Figura 11.

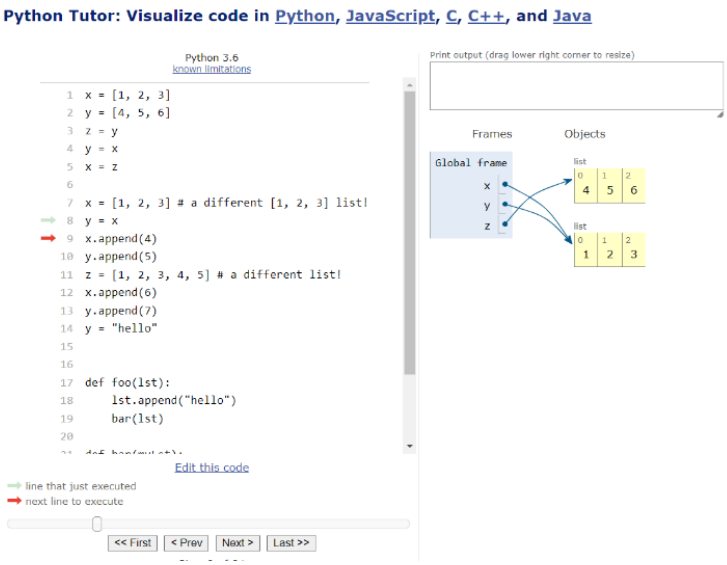

Figura 11. Ejecución paso a paso de un algoritmo en PythonTutor

Cabe destacar que de las herramientas presentes en las publicaciones analizadas solo PlanAni, PythonTutor y Jsvee están disponibles al público. Sin embargo, Jsvee y PlanAni no reciben mantenimiento activo desde hace años, lo cual puede limitar su eficacia y accesibilidad en el ámbito educativo actual.

III) Detección y prevención del plagio

La detección y prevención del plagio es un componente esencial en los cursos de programación, especialmente en entornos educativos donde se busca evaluar la comprensión auténtica de los estudiantes. En la enseñanza de programación, la facilidad para copiar código y la abundancia de soluciones en línea aumentan el riesgo de prácticas deshonestas, lo que hace fundamental contar con herramientas que puedan identificar similitudes en los trabajos entregados. Esta sección explora los casos de uso de herramientas de detección de plagio en programación, su implementación en cursos introductorios, y cómo contribuyen a fomentar la integridad académica, ofreciendo una retroalimentación oportuna y promoviendo prácticas éticas en el aprendizaje del código.

En las publicaciones analizadas, Kurniawan *et al.* [86], Karnalim *et al.* [27] y Lakshminarayanan y Rao [87] se enfocan en métodos educativos para prevenir el plagio. Estas publicaciones proponen el uso de herramientas que limiten el uso de los computadores durante evaluaciones BYOD [86], herramientas de retroalimentación formativa [27] y herramientas de control de versiones [87], promoviendo una conciencia ética y habilidades de autorregulación en los estudiantes.

[88], [89], [76] y [49] proponen y evalúan enfoques automáticos y semiautomáticos para detectar similitudes en el código, empleando una variedad de técnicas. Leonardo Mariani y Daniela Micucci [88] utilizan un análisis basado en tokens, mientras que Oscar Karnalim y Simon [25] combinan múltiples técnicas para identificar similitudes en segmentos de código ofreciendo una solución semi-automatizada a la detección de plagio. Finalmente, Bejarano *et al.* [89] proponen la herramienta CodeSight, la cual utiliza el algoritmo de *"Greedy string tiling"*, y Zhang *et al.* [49] implementan grafos de dependencia de programas (PDG) para detectar patrones de similitud en el

flujo de ejecución del código. Cada enfoque contribuye con metodologías específicas que mejoran la precisión y efectividad de los sistemas de detección de plagio en programación.

La mayoría de las herramientas de análisis de similitud en código presentan unas interfaces para (i) adjuntar los archivos a analizar, (2) comparar uno a uno los archivos y ver sus similitudes línea a línea y (iii) realizar un reporte de la similitud entre los archivos de manera porcentual. Estos sistemas se evidencian en la Figura 12 donde se presenta el resultado de un análisis de similitud de código usando CodeSight.

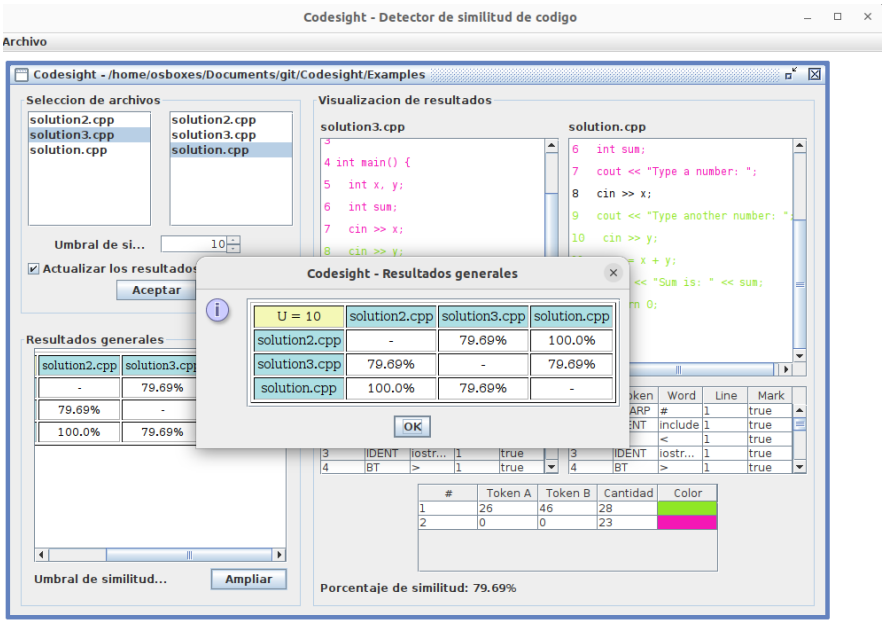

Figura 12. Resultado de un análisis de similitud en CodeSight

Entre otras publicaciones relacionadas con el plagio se encuentran una realizada por Butler *et al.* [90] donde examinan cómo variaciones en las preguntas de programación pueden reducir las posibilidades de plagio, explorando cómo la redacción y el diseño de tareas influyen en el comportamiento de los estudiantes y en la originalidad de sus respuestas; y otra realizada por Stella Biderman y Edward Raf [77], quienes investigan cómo los modelos de lenguaje como ChatGPT pueden manipularse para evadir sistemas de detección de plagio, usando MOSS como ejemplo, lo que destaca la necesidad de desarrollar herramientas más robustas para enfrentar estos desafíos. La Figura 13 presenta un análisis de similitud en MOSS, el cual se puede acceder por medio de una API y es un servicio ofrecido por la universidad de Stanford de manera gratuita, de uso bastante común en entornos académicos, que permite realizar análisis de similitud entre archivos y que brilla por su amplia cantidad de lenguajes soportados.

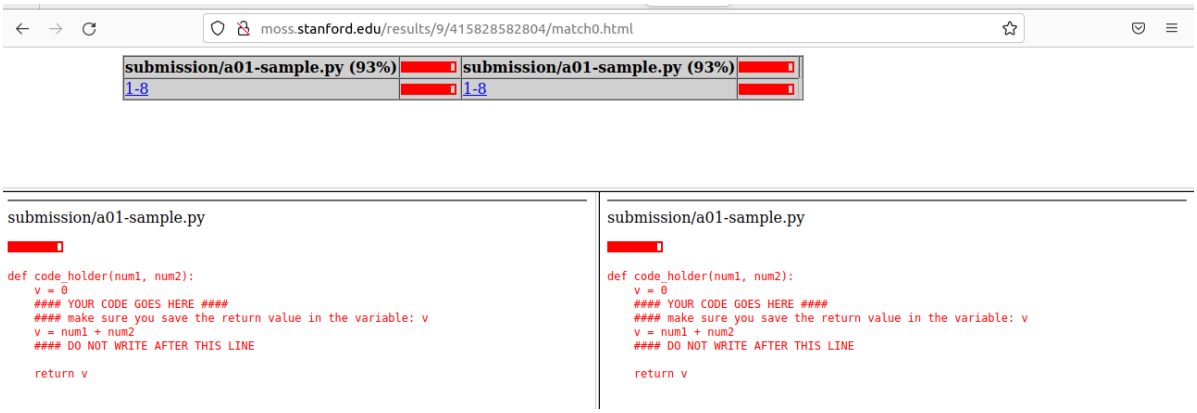

Figura 13. Resultado de un análisis de similitud en MOSS

En total, de las publicaciones, hay 7 que se enfocan en la detección del plagio y 3 que se centran en la prevención del plagio. Esto puede indicar una tendencia en el ámbito académico hacia la implementación de herramientas y técnicas de detección como medida principal para abordar el problema del plagio en programación, poniendo esfuerzos menores en la prevención del mismo.

IV) Retroalimentación automática

La retroalimentación automática es una herramienta clave en la enseñanza de programación, ya que permite a los estudiantes recibir evaluaciones inmediatas sobre su desempeño, favoreciendo un aprendizaje autónomo y activo. Además, estas herramientas reducen significativamente la carga de gestión para el docente, automatizando la revisión de tareas y la identificación de errores comunes. Al ofrecer comentarios y sugerencias en tiempo real, la retroalimentación automática no solo mejora la comprensión de los conceptos de programación, sino que también optimiza el proceso educativo al facilitar una asistencia continua y oportuna. En esta sección, se exploran publicaciones sobre el uso de herramientas de retroalimentación automática en entornos educativos y sus beneficios tanto para estudiantes como para docentes.

- **Jueces:** Herramientas como CodeRunner y Jutge.org han demostrado ser efectivos para evaluar habilidades de programación en estudiantes mediante un sistema de corrección automática que permite la ejecución de código en respuesta a preguntas programadas en múltiples lenguajes. Lobb y Harlow [40], muestran la efectividad de la herramienta CodeRunner en el entorno Moodle, facilitando la evaluación en cursos introductorios. Del mismo modo, UVa Online Judge, analizado por Verdú *et al.* [34], ofrece una plataforma abierta para resolver problemas de programación, incentivando la práctica continua. Estas herramientas actúan como jueces en línea, evaluando la precisión del código y brindando una retroalimentación oportuna.
- **Plataformas de evaluación automática:** Entre las herramientas enfocadas en la retroalimentación continua y validación del correcto comportamiento de los programas de los estudiantes. HAAP [91] se destaca por su enfoque en la gamificación y la programación funcional en Haskell, proporcionando un entorno interactivo que evalúa tanto el correcto funcionamiento de los programas como métricas de calidad interna. Mumuki [92], es otra plataforma integral diseñada para instituciones educativas y gobiernos, ofreciendo un enfoque de enseñanza estructurado y retroalimentación detallada. VPL Plugin, discutido por Demaidi *et al.* [93], gestiona asignaciones de programación en Moodle, proporcionando evaluación automatizada y retroalimentación constante. La Figura 14 muestra el entorno de

programación en el plugin VPL para Moodle. Este diseño es representativo de la mayoría de aplicaciones web con soporte para evaluación automática: un área destinada a escribir el código, una opción para "enviar" el programa y una sección para visualizar los resultados de su ejecución.

- **Librerías y herramientas de pruebas unitarias:** Estas herramientas son esenciales en el desarrollo de buenas prácticas de programación. JUnit, una de las herramientas de pruebas más populares en Java, es valorada por su papel en el desarrollo guiado por pruebas (TDD) y se utiliza ampliamente en cursos de programación para enseñar el proceso de pruebas unitarias, como describen Baruque y Herrero [72]. QuickCheck [67], una librería de pruebas de propiedades para Haskell, facilita la generación automática de casos de prueba. Zougari *et al.* [73] también promueve la práctica de pruebas unitarias para el marco .NET, haciendo uso del lenguaje C# y la librería xUnit.
- **Herramientas de verificación y prueba avanzada:** Gao *et al.* Presentan KLEE [66], una herramienta de ejecución simbólica basada en LLVM que permite verificar el comportamiento del código frente a múltiples casos de prueba y que es muy popular en entornos industriales. Por su parte, Cardenas-Cobo *et al.* presentan LAV [70], una herramienta de pruebas de regresión para LLVM que permite evaluar la consistencia del código en distintas versiones.
- **Herramientas de análisis estático y verificación de código:** En esta categoría, PMD y Pylint ofrecen un análisis estático del código para identificar errores comunes y mejorar la calidad del código en lenguajes como Java y Python, respectivamente. Molnar *et al.* en [78], muestra la utilidad de PMD para encontrar errores de programación y mantener estándares de calidad, mientras que Pylint apoya la detección de errores específicos en Python, promoviendo la mejora continua del código.

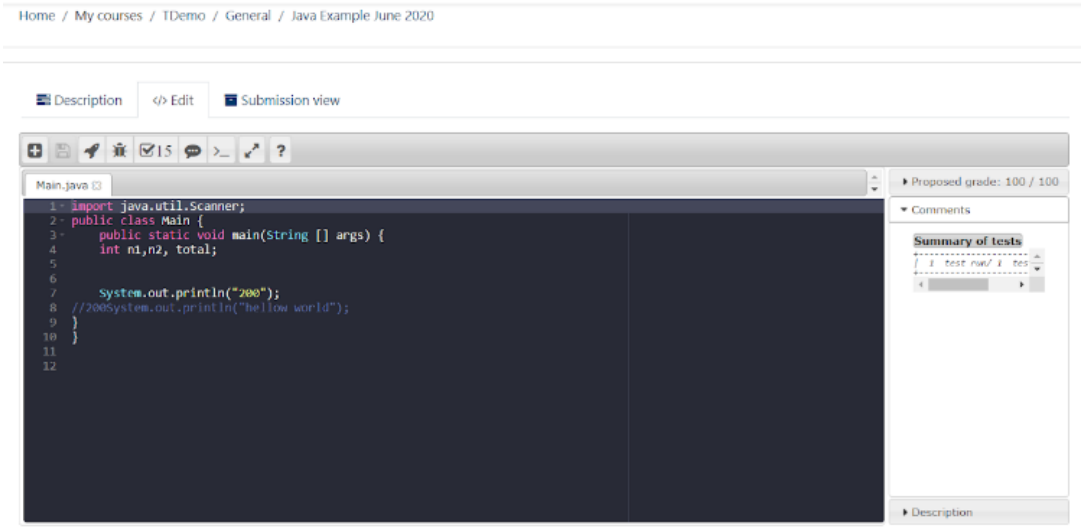

Figura 14. Solución de un ejercicio en la herramienta VPL Plugin para Moodle

La clasificación de la retroalimentación en programación según el tipo de interacción, definida por Hieke Keuning et. al en *"A Systematic Literature Review of Automated Feedback Generation for Programming Exercises"* [16], distingue entre retroalimentación dinámica, híbrida y estática. La retroalimentación dinámica se enfoca en evaluar la ejecución del código a partir de la entrada y salida, generalmente implementada a través de jueces automáticos, librerías de pruebas y sistemas

de calificación automatizados. La retroalimentación híbrida combina la evaluación de la ejecución con el análisis del código en tiempo real, mientras que la retroalimentación estática se basa únicamente en el análisis de código y proporciona comentarios inmediatos mientras el estudiante programa, sin necesidad de ejecutar el código.

La Figura 15 muestra la distribución de publicaciones en este estudio según el tipo de retroalimentación: 9 con retroalimentación dinámica, 6 con híbrida y 4 con estática. Es notable que la mayoría de las publicaciones se orientan a la retroalimentación dinámica, dado que las evaluaciones de entrada y salida mediante sistemas automáticos son el método más común en cursos de programación. En contraste, la retroalimentación estática es menos frecuente, aunque resulta útil para ofrecer comentarios en tiempo real que ayuden al estudiante a mejorar su código de manera autónoma y a prevenir o corregir errores básicos de programación.

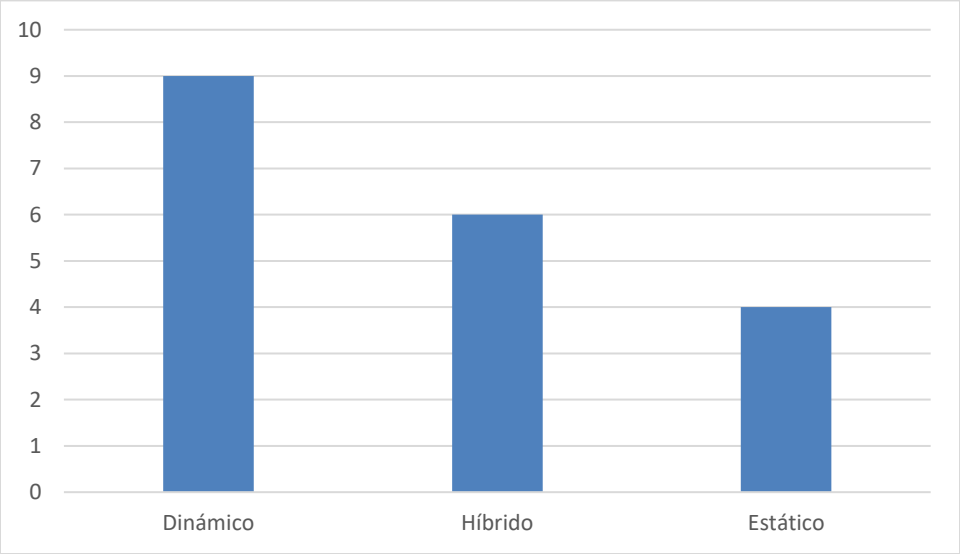

Figura 15. Distribución de las publicaciones de acuerdo a su tipo de retroalimentación

V) Asistencia a estudiantes

Las herramientas de asistencia para estudiantes en programación ofrecen soporte continuo durante el aprendizaje, agrupándose en cuatro subcategorías: (i) sugerencias automáticas, que guían a los estudiantes en tiempo real optimizando su código y proporcionando retroalimentación inmediata; (ii) entornos de desarrollo integrados (IDE), con funcionalidades como corrección automática y visualización de errores; (iii) plataformas de colaboración social, que permiten la interacción en comunidad y la resolución de dudas colectivas; y (iv) chatbots, que responden preguntas de manera personalizada y orientan a los estudiantes en la resolución de problemas específicos.

Las investigaciones sobre sugerencias automáticas se centran en mejorar el aprendizaje en programación a través de retroalimentación adaptativa y asistencia automatizada. Marwan et. La hallaron que al añadir explicaciones textuales a las sugerencias de "siguiente paso", los estudiantes las percibieron como más útiles, aunque no siempre mejoraron su rendimiento [31]. Feldman *et al.* [57] utilizaron la síntesis de programas para crear una herramienta que responde automáticamente a preguntas sobre el progreso de los estudiantes en tiempo real. Otros trabajos abordaron un sistema de ayuda que ajusta la frecuencia de sugerencias según el comportamiento del estudiante [30] y

narraciones textuales para mejorar la comprensión de algoritmos [44]. En entornos adaptativos, recomendaron ejercicios de Java o Scratch para mejorar la motivación y el desempeño académico [42], [69].

Por otro lado, varios estudios desarrollaron IDEs en línea optimizados para el aprendizaje de programación. En un estudio realizado por Alharbi [94], la transición de un laboratorio tradicional a un IDE web mejoró las percepciones de accesibilidad de los estudiantes. En cuando a la colaboración, Ghorashi y Jensen presentan Jimbo [52], un IDE que permite el trabajo en equipo en tiempo real a través de la web. Otros entornos como WIDE [59] y ViPLab [62] ofrecen plataformas para que los estudiantes de C programen y se autoevalúen. También, como alternativa para uso en vivo en el aula, Alammary *et al.* [63] presentan un "Smart Lab" que ayuda a los instructores a monitorear el progreso de los estudiantes y brindar retroalimentación oportuna en la resolución de errores.

Las herramientas de colaboración social también destacan en el apoyo grupal al aprendizaje en programación. EduCo [48] integra gestión de proyectos y redes sociales, promoviendo la colaboración entre estudiantes e instructores. Un sistema de groupware para aprendizaje colaborativo permite que estudiantes y tutores resuelvan problemas de programación en un espacio compartido [54].

Para apoyo individualizado, Python-Bot [75] y Revision-Bot [71] asisten a los estudiantes en conceptos y prácticas de exámenes mediante chatbots, mostrando efectividad en el aprendizaje de Python.

La Figura 10 muestra la distribución de publicaciones según la subcategoría de herramientas de asistencia estudiadas. Con un total de 19 publicaciones, las sugerencias automáticas representan la subcategoría más explorada (8 publicaciones), seguida por los IDE (7 publicaciones). Las subcategorías de colaboración social y chatbots cuentan con 2 publicaciones cada una. Este desglose revela un mayor interés en las sugerencias y los IDE, que ofrecen asistencia inmediata y funcionalidades avanzadas, mientras que la colaboración social y los chatbots, aunque útiles, aparecen menos estudiados en la literatura.

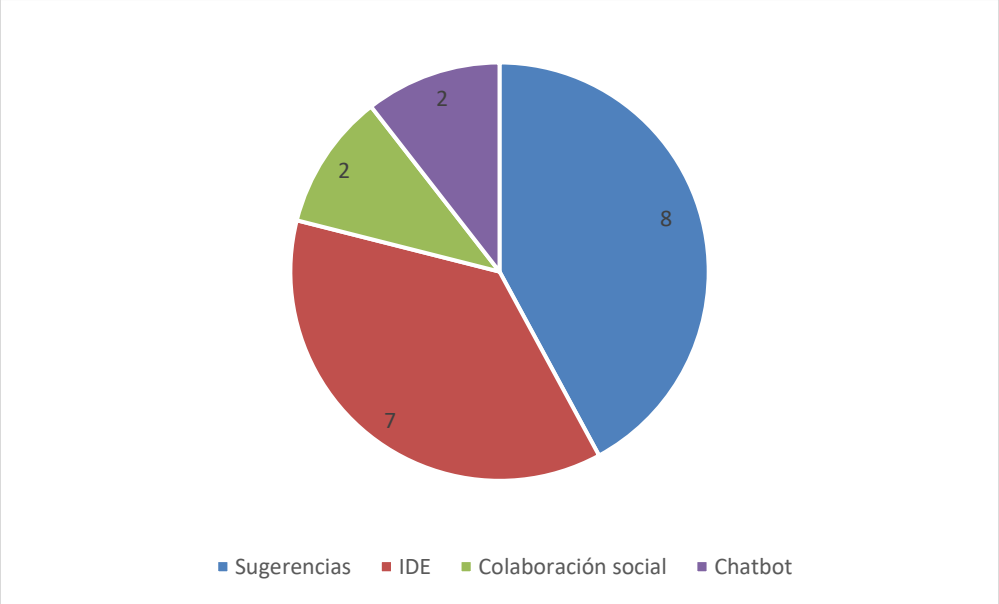

Figura 16. Distribución de publicaciones según la subcategoría de herramientas de asistencia

VI) Implicación

La motivación y la implicación de los estudiantes en su proceso de aprendizaje son fundamentales para mejorar la efectividad en la enseñanza de programación introductoria. Según Fredricks *et al.* [95], esta implicación se manifiesta en tres dimensiones listadas en la Tabla 13. La dimensión conductual, que surge de la participación obligada o motivada por la metodología; la dimensión cognitiva, impulsada por el interés, la curiosidad y el deseo de comprender; y la dimensión afectiva, vinculada a emociones positivas hacia el aprendizaje. Estas dimensiones, en conjunto, fomentan un aprendizaje más profundo y sostenido, esencial para el éxito en programación.

Estrategia	**Descripción**	**Mecanismos utilizados**
Conductual	Fomenta la participación activa mediante la motivación de los estudiantes	Retroalimentación, sugerencias, límites de tiempo
Cognitiva	Captura el interés de los estudiantes y estimula la comprensión profunda en temas específicos	Desarrollo de juegos, robótica, nuevas tecnologías
Afectiva	Genera una conexión emocional positiva para un aprendizaje más agradable y sostenido	Gamificación

Tabla 13. Estrategias utilizadas para motivar a los estudiantes en cursos de programación

En esta sección, analizamos publicaciones que emplean técnicas como la gamificación, la robótica o el desarrollo de juegos, así como estrategias diseñadas para captar y mantener el interés de los estudiantes, aumentando el atractivo de los cursos y su implicación en ellos.

Las investigaciones centradas en la implicación conductual han explorado diversas estrategias para mejorar la participación activa de los estudiantes. Marwan *et al.* (2020) [32] examinaron la efectividad de la retroalimentación adaptativa en tiempo real, mostrando que esta mejora la intención de los estudiantes de persistir en programación al proporcionarles comentarios inmediatos y personalizados. Por otro lado, Denny *et al.* (2021) [29] destacaron el uso de la gamificación y sistemas de retroalimentación programada como métodos efectivos para fomentar una mayor interacción y gestión del tiempo en los estudiantes, incentivando el inicio temprano de las tareas y la participación constante. Además, Nelson *et al.* (2017) [96] investigaron herramientas como ACP, que permiten a los estudiantes practicar activamente dentro y fuera del aula mientras son guiados en el aprendizaje de nuevos conceptos de programación.

En cuanto a la implicación cognitiva, Yoshimura *et al.* (2022) [60] estudiaron sistemas de recomendación que proporcionan problemas relacionados con las tareas actuales, ayudando a los estudiantes a fortalecer su comprensión de conceptos clave. Adicionalmente, Anderson y Gavan (2012) [79] exploraron el uso de robots y juegos de simulación para la enseñanza de conceptos de programación y pensamiento abstracto, concluyendo que estas herramientas mejoran significativamente el desarrollo de habilidades cognitivas en los estudiantes. Pérez-Marín *et al.* (2014) [50] también subrayaron cómo los entornos de desarrollo de videojuegos y simulaciones pueden mejorar la comprensión de conceptos abstractos, haciendo que la experiencia de aprendizaje sea más atractiva y efectiva.

Las investigaciones relacionadas con la implicación afectiva combinan elementos conductuales y afectivos mediante el uso de tecnologías llamativas como la gamificación y la realidad aumentada, con el propósito de mejorar tanto la participación activa de los estudiantes como su disfrute del aprendizaje. Denny *et al.* (2021) [24] y Khaleel *et al.* (2019) [45] analizaron sistemas de gamificación aplicados a cursos de programación, mostrando que estos motivan a los estudiantes a involucrarse activamente en sus tareas. Por otro lado, Ou Yang *et al.* (2023) [47] evaluaron herramientas de realidad aumentada y entornos de aprendizaje inmersivos, destacando cómo estas tecnologías aumentan la satisfacción y percepción positiva hacia el aprendizaje. Además, Mathrani *et al.* (2016) [38] investigaron el impacto de PlayIT, un sistema que incorpora elementos de juego para facilitar la comprensión de conceptos complejos, concluyendo que su uso mejora la implicación de los estudiantes y su desempeño en las evaluaciones.

La Figura 17 muestra los métodos de implicación utilizados en las distintas publicaciones. De un total de 22 publicaciones, 17 emplearon la implicación conductual, 14 la afectiva y 6 la cognitiva. Esta distribución sugiere un mayor interés en aplicar técnicas que motiven o exijan a los estudiantes a trabajar activamente, mientras se presta menor atención a hacer los temas enseñados más atractivos e interesantes. Esto podría reflejar una tendencia a priorizar métodos que promuevan la disciplina y la constancia sobre aquellos que estimulen la curiosidad y el interés profundo en los contenidos de los cursos.

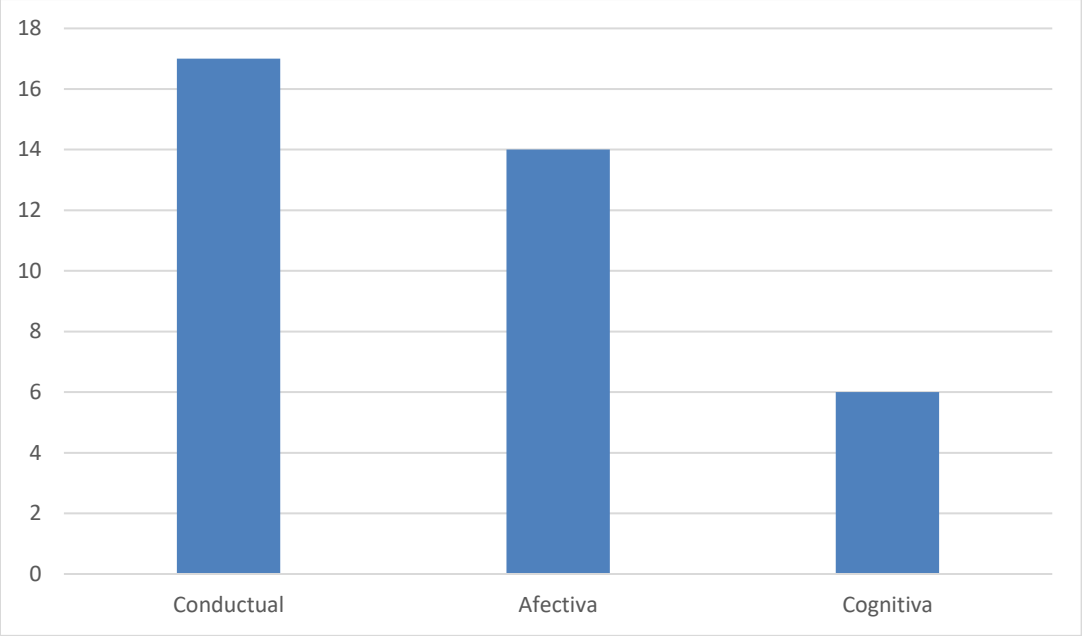

Figura 17. Métodos de implicación utilizados en los cursos

4.2.5 Tendencias y retos

En esta sección respondemos a PI4, abordando las principales tendencias y retos asociados al uso de laboratorios virtuales de programación en la educación basados en nuestros hallazgos en la literatura. Estos entornos han ganado popularidad por su accesibilidad y flexibilidad, facilitando la práctica de programación en una variedad de contextos educativos. Sin embargo, su implementación presenta desafíos, como la necesidad de infraestructura tecnológica adecuada y el desarrollo de métodos de evaluación y retroalimentación efectivos. A continuación, exploramos tanto las tendencias que impulsan el uso de laboratorios virtuales como los obstáculos que limitan su efectividad en el aprendizaje de programación.

I) Programación visual

La programación visual continúa siendo una tendencia atractiva en la educación, especialmente en la enseñanza de programación introductoria. Los lenguajes de programación basados en bloques, como Scratch y Alice, han demostrado ser especialmente efectivos para captar la atención de los estudiantes, facilitando la comprensión de conceptos básicos sin las barreras de la sintaxis tradicional [35], [37]. Aunque existe un interés en estos métodos, no se ha observado un aumento significativo en el número de publicaciones que empleen lenguajes de programación gráficos, sugiriendo que, aunque la programación visual es llamativa, su adopción sigue siendo limitada en el ámbito académico.

A pesar de sus ventajas, la programación visual enfrenta desafíos importantes. Existen solo tres lenguajes de programación visuales ampliamente utilizados, y otros enfoques, como los diagramas de flujo o UML, son menos populares en comparación con los lenguajes basados en bloques, que dominan este campo. Esta limitación abre una oportunidad para el desarrollo de herramientas mixtas que integren diferentes tipos de representación visual, como diagramas de flujo o bloques y UML, ofreciendo una experiencia de aprendizaje más completa y diversificada.

II) Visualización de algoritmos

Las visualizaciones de algoritmos están ganando relevancia en la educación de programación, permitiendo que los estudiantes comprendan conceptos complejos de una manera más intuitiva. Herramientas como PlanAni han demostrado ser efectivas para enseñar programación en lenguajes como C/C++, donde la visualización de punteros es fundamental para entender estructuras de datos complejas y operaciones en memoria [85]. La implementación de estas herramientas permite a los estudiantes mejorar en cursos avanzados al facilitar el aprendizaje de operaciones y estructuras que, de otro modo, podrían resultar abstractas y difíciles de conceptualizar.

Sin embargo, estas herramientas enfrentan retos importantes en su implementación y uso pedagógico. En muchos casos, los sistemas de visualización actuales funcionan como "animaciones" o "películas" que los estudiantes observan sin interacción activa, limitando su capacidad para practicar y experimentar directamente con el código. En un estudio, solo 4 de las 6 herramientas estudiadas permitían que los estudiantes programaran activamente, mientras que las otras se usaban exclusivamente para explicaciones y demostraciones en clase [84]. Esto plantea un desafío en la creación de herramientas de visualización que permitan a los estudiantes interactuar, experimentar y practicar de manera efectiva, especialmente en cursos donde se busca que la visualización sirva de soporte para la ejecución y el aprendizaje activo.

El desarrollo de herramientas de visualización que integren tanto la programación como la visualización gráfica en tiempo real representa un reto interesante. Algunas herramientas, como PythonTutor, han mostrado un impacto positivo en el rendimiento de los estudiantes, permitiéndoles experimentar directamente con el código y ver el flujo de ejecución y cambios en la memoria en tiempo real [28]. Sin embargo, otras investigaciones indican que una visualización tan simplificada puede no cubrir adecuadamente las necesidades de cursos más avanzados o de conceptos más complejos, como la gestión de estructuras de datos en distintos lenguajes [26].

La creación de una herramienta de visualización que sea lo suficientemente robusta y flexible para adaptarse a múltiples lenguajes y que permita a los estudiantes programar y visualizar al mismo tiempo, sigue siendo un área con gran potencial en la educación de programación. C Tutor, por ejemplo, ha demostrado ser útil en cursos de introducción a la programación, mostrando mejoras en la autoconfianza y logros de los estudiantes en Portugal y Serbia [81]. Sin embargo, herramientas como VisuAlgo [97] y otros sistemas comerciales de visualización, que también resultan populares en la industria, tienden a centrarse en el usuario observador más que en el usuario activo, destacando la necesidad de seguir innovando en la creación de herramientas que puedan servir de puente entre la visualización y la práctica activa en programación.

III) Detección y prevención del plagio

En el ámbito de la programación educativa, existe una tendencia marcada a identificar la similitud de código en lugar de centrarse en la prevención activa del plagio. Diversos estudios han explorado técnicas avanzadas para analizar la similitud del código a través de gráficos de ejecución, permitiendo una comparación estructural que va más allá del simple análisis textual [49]. Este enfoque permite identificar similitudes en la ejecución y lógica del código, incluso cuando los estudiantes aplican técnicas de obfuscación. Otro método en crecimiento es el uso de una "solución de referencia", un código modelo al que las soluciones de los estudiantes pueden asemejarse estructuralmente sin que se considere plagio, permitiendo así un margen para la creatividad en el

código de los estudiantes [88]. Entre las técnicas comunes de detección, los métodos como *"greedy-string tiling"* y el análisis basado en tokens son los más utilizados. Sin embargo, muchas instituciones y estudios recurren a servicios externos como MOSS para simplificar el proceso de detección.

A pesar de estos avances en la identificación de similitud de código, los esfuerzos en la prevención del plagio y en el fomento de prácticas académicas éticas son aún limitados. Solo en contadas ocasiones se investigan mecanismos de prevención efectivos, como el uso de sistemas de control de versiones (e.g., Git) para verificar el desarrollo histórico de los proyectos y asegurar que el trabajo haya sido realizado progresivamente por el estudiante [87]. Otro ejemplo es el empleo de tecnologías de sandboxing en exámenes BYOD (Bring Your Own Device), lo cual permite que los estudiantes usen sus dispositivos personales en un entorno controlado que restringe el acceso a recursos externos [86].

Una oportunidad de mejora radica en el desarrollo de algoritmos capaces de identificar patrones comunes de código que no constituyan plagio, sin necesidad de intervención manual o procesos semi-automatizados. Esto reduciría los falsos positivos y mejoraría la precisión en la detección de similitud. Asimismo, enfatizar en estrategias preventivas y formativas, como proporcionar retroalimentación sobre prácticas éticas, podría disminuir la dependencia en herramientas de detección y fomentar un enfoque proactivo en la educación en programación. Esta área de desarrollo representa un desafío importante, dado que requiere una infraestructura que no solo detecte plagio, sino que también permita a los estudiantes adquirir prácticas académicas íntegras desde el inicio de su formación.

IV) Retroalimentación automática

La retroalimentación automática en programación sigue una tendencia marcada hacia sistemas de retroalimentación dinámica o híbrida, empleando principalmente herramientas de autoevaluación, jueces automáticos y librerías de pruebas unitarias. En particular, los jueces automáticos, aunque cuentan con buen soporte para múltiples lenguajes, no han ganado gran popularidad debido a la limitada calidad de su retroalimentación, que usualmente se limita a señalar si el resultado es correcto o incorrecto sin detallar los errores específicos ni guiar al estudiante en su resolución [40], [98]. Esta carencia ha impulsado el interés en herramientas de retroalimentación estática que van más allá de la verificación de la respuesta, ofreciendo análisis de errores comunes, como problemas de sintaxis o lógica, y mejorando así el aprendizaje de programación en un entorno más instructivo [66], [80].

Uno de los retos principales es la complejidad del uso de jueces automáticos en entornos educativos. Aunque son herramientas de retroalimentación dinámica con soporte para diversos lenguajes de programación, presentan barreras de entrada debido a la falta de claridad en los mensajes de error y la limitada visibilidad de los casos de prueba en que se falla, como en el UVa Online Judge. Esto limita su efectividad para el aprendizaje, y un juez que supere estos problemas mediante una retroalimentación detallada y accesible sería un avance importante para su adopción en el ámbito académico [98].

Por otro lado, las librerías de pruebas unitarias, aunque menos populares en entornos educativos debido a los requerimientos de instalación y configuración local, presentan una oportunidad interesante. Si bien estos requisitos pueden ser un obstáculo para estudiantes principiantes, una

estrategia de implementación adecuada podría reducir significativamente los costos de mantenimiento y diseño para los docentes, ya que prácticamente no requieren infraestructura adicional. Este ahorro en infraestructura es particularmente valioso para cursos con grandes cantidades de estudiantes, donde los recursos disponibles para el mantenimiento de herramientas web pueden ser limitados.

Finalmente, la limitada disponibilidad de herramientas de retroalimentación estática fuera de entornos de desarrollo integrados (IDE) sugiere un área de oportunidad importante. El desarrollo de herramientas de retroalimentación estática efectivas permitiría una evaluación más profunda de errores comunes y un apoyo a la comprensión autónoma de los estudiantes, ofreciendo retroalimentación instructiva y personalizada, lo cual potenciaría la efectividad de la enseñanza de programación [78].

V) Asistencia a estudiantes

El uso de herramientas que proporcionan retroalimentación automática en tiempo real es una tendencia clave en la asistencia a estudiantes de programación. Estas herramientas, como algunos entornos de desarrollo integrados (IDEs) que incluyen sistemas de corrección de errores y sugerencias, ayudan a los estudiantes a identificar y solucionar problemas de sintaxis y estilo de forma instantánea, promoviendo un aprendizaje incremental y continuo [57]. Otra tendencia relevante es la integración de pistas adaptativas que desbloquean el progreso del estudiante al enfrentarse a bloqueos en su código; estas pistas permiten una comprensión más profunda al contextualizar los errores y guiar al estudiante hacia decisiones informadas [31]. Asimismo, el COVID-19 impulsó el desarrollo de herramientas colaborativas que permiten a los estudiantes trabajar juntos en proyectos de programación en tiempo real. IDEs colaborativos, como Jimbo [52], facilitan la comunicación y el aprendizaje conjunto, simulando escenarios de trabajo en equipo propios de la industria. Adicionalmente, la personalización y gamificación también son tendencias que buscan aumentar la motivación de los estudiantes; herramientas como Python-Bot [75] y Revision-Bot [71] ofrecen experiencias de aprendizaje adaptadas y atractivas, mientras que el sistema CARAMBA [69] en Scratch permite que los ejercicios se ajusten al nivel y ritmo del estudiante, incentivando un compromiso sostenido.

Uno de los principales retos en la implementación de herramientas de retroalimentación automática es la tendencia de los estudiantes a depender excesivamente de la ayuda constante, un fenómeno conocido como comportamiento de búsqueda de ayuda no productiva. Este comportamiento plantea preocupaciones sobre la efectividad de la retroalimentación rápida para fomentar un aprendizaje profundo, ya que podría inhibir la capacidad de los estudiantes para resolver problemas de manera autónoma [30]. Los IDEs también presentan el desafío de emitir mensajes de error que, al ser demasiado técnicos, resultan incomprensibles para estudiantes novatos, limitando su capacidad para aprender de los errores de manera efectiva. Además, aunque los sistemas de pistas y asistencia han mostrado resultados prometedores, muchos de ellos son aún pruebas de concepto y no están implementados a gran escala. Sin embargo, estos estudios indican una gran oportunidad para mejorar los cursos de programación mediante herramientas que impulsen la autonomía del estudiante y su capacidad de resolver dudas por sí mismo [57].

VI) Implicación y motivación

En el ámbito de la implicación en educación en programación, la mayoría de las intervenciones se concentran en la implicación conductual, que se enfoca en mantener la participación activa de los

estudiantes mediante estrategias como la retroalimentación inmediata y la estructuración de actividades. Un ejemplo destacado es el uso de retroalimentación inmediata adaptativa, que ha demostrado aumentar la intención de los estudiantes de continuar en la programación y reducir los tiempos de inactividad, incrementando así el compromiso con la tarea [32]. Otro enfoque conductual incluye la gamificación y la planificación de entregas con retroalimentación automática, lo cual incentiva a los estudiantes a comenzar temprano y a mantener un ritmo de trabajo constante, beneficiando especialmente a aquellos con mayor riesgo de bajo rendimiento [29].

En contraste, los métodos de implicación cognitiva, aunque menos frecuentes, emplean temas de interés como el desarrollo de videojuegos y la robótica para captar la atención de los estudiantes y motivarlos a aprender. Por ejemplo, la robótica se utiliza para enseñar conceptos de programación y pensamiento abstracto, logrando un impacto positivo en el desarrollo de habilidades cognitivas [99], [79]. Otro enfoque cognitivo efectivo es la sugerencia de problemas similares a los ya resueltos, lo cual fomenta la curiosidad y ayuda a consolidar la comprensión de los conceptos de programación [60]. Estos métodos no solo estimulan el aprendizaje activo, sino que también promueven una reflexión más profunda y significativa sobre los contenidos.

La implicación afectiva, que se encuentra menos explorada en la literatura, representa un área con un gran potencial de investigación. Este tipo de implicación busca mejorar la satisfacción y el disfrute en el aprendizaje mediante el uso de tecnologías inmersivas como la realidad aumentada [47] y los entornos de gamificación [24]. Estas herramientas han mostrado ser efectivas para aumentar el disfrute y la percepción positiva del aprendizaje en programación, generando un entorno de estudio más atractivo y motivador. Un ejemplo relevante es el entorno de realidad virtual iProgVR [51], que aborda problemas comunes en la enseñanza de programación, como la abstracción y los conceptos erróneos, al presentar estos conceptos de forma visual e interactiva, mejorando significativamente la comprensión y motivación de los estudiantes.

En resumen, aunque se observa una tendencia predominante hacia métodos de implicación conductual, existe un espacio amplio para explorar y expandir el uso de estrategias afectivas y cognitivas que fomenten no solo la participación activa, sino también el disfrute y la curiosidad en los estudiantes. Este enfoque integral podría aportar a un aprendizaje más significativo y sostenido en el tiempo.

VII Aspectos técnicos y contextuales

El desarrollo de herramientas gratuitas y de código abierto ha ganado popularidad en el ámbito educativo, facilitando un acceso más amplio y económico a recursos destinados a la enseñanza de la programación. Además, se observa una creciente preferencia por las aplicaciones web en lugar de aquellas que requieren instalación local en los dispositivos de las instituciones o de los estudiantes. Las aplicaciones web ofrecen mayor accesibilidad y eliminan problemas de configuración, una ventaja particularmente relevante durante la pandemia de COVID-19, cuando el aprendizaje remoto se convirtió en una necesidad. Aunque no se sabe si esta tendencia sigue vigente, su impacto en la adopción de herramientas en línea ha sido significativo. Por otra parte, muchas herramientas de programación se integran en Moodle, una plataforma de gestión de aprendizaje de código abierto. La infraestructura flexible de Moodle permite que estas herramientas extiendan su funcionalidad y puedan cubrir más áreas de aprendizaje, como la retroalimentación automatizada, más allá de la autoevaluación básica.

Sin embargo, el desarrollo de herramientas educativas enfrenta el reto de su corta vida útil. En muchos casos, estas herramientas se limitan a pruebas de concepto o proyectos académicos que, tras su publicación, quedan abandonados, lo cual limita su uso en entornos de aprendizaje permanentes. A pesar de que muchas son de código abierto, carecen de un soporte o documentación, lo cual reduce significativamente su aplicabilidad. Otro desafío es la escasa representación de los lenguajes de programación funcionales en estas herramientas, lo cual limita las opciones para docentes y estudiantes interesados en esta área, quedando en desventaja frente a los lenguajes de programación más populares, como son Python, Java, C y C++.

4.3 Discusión

En este trabajo se analizaron las publicaciones relacionadas con el uso de laboratorios virtuales de programación, explorando sus aplicaciones, ventajas y limitaciones. Este análisis permitió identificar aspectos clave asociados al uso de herramientas en la enseñanza de programación, proporcionando una visión integral de su impacto en el ámbito educativo. A continuación, se presentan los hallazgos organizados por áreas de interés, contrastando y complementando nuestros resultados con los de estudios previos.

Autores, países y frecuencias de las publicaciones

El análisis demográfico de las publicaciones en este estudio proporciona una visión amplia de las tendencias en investigación sobre herramientas para la enseñanza de programación introductoria. En cuanto a los países, Estados Unidos lidera con un 21% de las publicaciones, seguido por España, Nueva Zelanda, Finlandia y Sudáfrica. Estos 5 países en conjunto representan el 44% del total de publicaciones. Este predominio de ciertos países destaca las diferencias en la producción académica global, mientras que la participación de otros 35 países evidencia una diversidad en el interés por este campo. Además, el análisis de autores muestra patrones de colaboración significativos, con figuras influyentes como Simon y Andrew Luxton-Reilly, quienes actúan como puentes entre grupos de investigación.

En términos de distribución temporal, el estudio identifica picos en los años 2016, 2018, 2020 y 2022, siendo este último el más productivo con 15 publicaciones. El aumento en 2020 puede asociarse a la pandemia de COVID-19, que probablemente incentivó la investigación en educación remota. Las publicaciones más citadas revelan que las categorías predominantes son “retroalimentación automática” y “programación visual”, representando 5 de las 10 más citadas, lo que destaca su relevancia para mejorar la enseñanza de programación al proporcionar evaluaciones inmediatas y entornos que facilitan la comprensión conceptual. Sin embargo, es notable que la categoría de "plagio" no esté presente en el top 10, a pesar de su importancia en la evaluación académica. Esto podría indicar que la detección de plagio, aunque relevante, no ha sido un foco principal en los estudios más influyentes, posiblemente porque se considera un problema ya abordado por herramientas existentes o porque los investigadores han priorizado áreas más innovadoras como la retroalimentación y la visualización.

Este análisis muestra un campo de investigación dinámico con una distribución desigual pero creciente en términos geográficos, temporales y temáticos. Las colaboraciones entre autores y los picos temporales reflejan el interés constante por mejorar la enseñanza de programación, mientras

que las citaciones destacan áreas clave como la retroalimentación y la programación visual, esenciales para facilitar el aprendizaje en estudiantes de nivel introductorio.

Lenguajes y herramientas

El análisis de los lenguajes de programación empleados en las publicaciones revela una clara preferencia por Java, Python, C y C++, que lideran en popularidad, destacando los distintos enfoques educativos que representan. Mientras que C suele enfocarse en la algoritmia imperativa básica, C++ se orienta hacia la enseñanza de la programación orientada a objetos (POO), el paradigma más representado en el estudio con 45 publicaciones. Python, que ha mostrado un notable incremento desde 2018, especialmente en 2022, sobresale por su practicidad y su creciente adopción en contextos educativos e investigativos, consolidándose como un lenguaje accesible y versátil. Por otro lado, los lenguajes gráficos como Scratch e iSnap, aunque menos representados, juegan un papel importante en la introducción a conceptos básicos de programación, especialmente en el ámbito visual. En contraste, los lenguajes funcionales como Haskell y Lisp tienen una representación marginal, con solo 3 publicaciones, reflejando un interés significativamente menor en este paradigma dentro de los estudios analizados. Esto sugiere que, aunque los paradigmas imperativos y orientados a objetos dominan, el uso de lenguajes visuales y funcionales podría ser un área de oportunidad para diversificar las estrategias educativas en programación.

El análisis de las herramientas de programación utilizadas en las publicaciones revela una amplia diversidad en cuanto a propósitos y funcionalidades. Las herramientas más destacadas incluyen Scratch, MOSS, iSnap, Alice, PythonTutor y JUnit, cada una con múltiples menciones en las publicaciones revisadas. Estas herramientas abordan necesidades específicas en contextos educativos, como la programación visual, la detección de plagio, la visualización de algoritmos y la automatización de pruebas. Además, la mayoría de estas herramientas son gratuitas, cuentan con soporte activo y una documentación adecuada, características que facilitan su adopción en entornos educativos. En contraste, 47 herramientas fueron descartadas debido a su falta de disponibilidad pública, incompatibilidad técnica o dependencia de servicios obsoletos, lo que evidencia las limitaciones que enfrentan algunas tecnologías al intentar integrarse en investigaciones educativas.

Entre las herramientas destacadas, se observa una fuerte preferencia por aquellas que ofrecen funcionalidades de evaluación automática, como jueces de código y librerías de pruebas. Estas herramientas, junto con las de visualización de código y programación visual, son ampliamente valoradas por su capacidad de proporcionar retroalimentación inmediata y facilitar el aprendizaje práctico. Además, aunque las herramientas de código abierto predominan por su accesibilidad y adaptabilidad, algunas herramientas propietarias, como Replit y MOSS, sobresalen debido a su calidad, documentación y soporte. Esto sugiere que la combinación de herramientas accesibles y propietarias puede ser una estrategia efectiva para satisfacer las necesidades educativas en cursos de introducción a la programación, permitiendo un balance entre innovación y practicidad.

Fines y fases del proceso educativo

El análisis de los fines y fases del uso de laboratorios virtuales de programación revela un amplio rango de aplicaciones que abordan distintas fases del aprendizaje en cursos introductorios. La programación visual y la visualización de algoritmos destacan por su capacidad de simplificar conceptos abstractos y reducir barreras iniciales, haciendo que los estudiantes se enfoquen en la lógica y estructura de la programación sin preocuparse inicialmente por la sintaxis. Estas herramientas no solo mejoran la comprensión conceptual, sino que también fomentan la motivación

al hacer el aprendizaje más accesible e interactivo. Ejemplos como Scratch, Alice o PythonTutor demuestran cómo estas estrategias promueven un entendimiento profundo mediante representaciones gráficas y ejecuciones paso a paso. Sin embargo, la falta de mantenimiento activo de la mayoría de las herramientas relacionadas limita su eficacia y accesibilidad en contextos educativos modernos.

La detección y prevención del plagio emergen como componentes esenciales en estos laboratorios, especialmente en cursos donde la evaluación de la autoría es crítica para el aprendizaje auténtico. Herramientas como MOSS y CodeSight han sido fundamentales para identificar similitudes en el código, mientras que enfoques como la variación de tareas y el control de versiones buscan prevenir directamente el plagio. A pesar de su importancia, el enfoque dominante hacia la detección más que la prevención sugiere que aún hay espacio para innovar en metodologías que promuevan una ética de programación proactiva desde las etapas iniciales del aprendizaje.

La retroalimentación automática y las herramientas de asistencia para estudiantes complementan estas estrategias al ofrecer evaluaciones inmediatas y apoyo continuo. Jueces automáticos, librerías de pruebas y plataformas como CodeRunner y Jutge.org facilitan el aprendizaje autónomo y reducen la carga del docente al automatizar tareas repetitivas. Simultáneamente, herramientas de asistencia, como chatbots y entornos colaborativos, promueven una experiencia de aprendizaje personalizada y comunitaria, aunque su presencia en la literatura es menos destacada. Estas herramientas refuerzan la importancia de integrar la tecnología para atender tanto las necesidades técnicas como pedagógicas en cursos introductorios de programación.

Finalmente, la implicación de los estudiantes se aborda mediante estrategias como la gamificación, el uso de robots y simulaciones, y tecnologías inmersivas como la realidad aumentada. Estas técnicas no solo aumentan la participación activa, sino que también influyen positivamente en el disfrute del aprendizaje, fomentando una relación más significativa con los contenidos. Sin embargo, el mayor enfoque en la implicación conductual sugiere que las metodologías aún priorizan la disciplina y la constancia sobre el interés intrínseco, dejando abierta la oportunidad de explorar métodos que hagan los cursos más atractivos y emocionalmente enriquecedores.

Tendencias y retos

El análisis de tendencias y retos en el uso de laboratorios virtuales de programación muestra un panorama de creciente interés y adopción, pero también subraya desafíos importantes. La programación visual sigue siendo una estrategia educativa destacada, especialmente para estudiantes principiantes, al eliminar barreras de entrada relacionadas con la sintaxis y enfocarse en la lógica. Sin embargo, la limitada diversidad de lenguajes visuales disponibles y su adopción moderada sugieren oportunidades para desarrollar herramientas que combinen enfoques visuales, como bloques y diagramas de flujo, con métodos tradicionales para enriquecer el aprendizaje. Paralelamente, la visualización de algoritmos está ganando relevancia al facilitar la comprensión de conceptos complejos, aunque persisten desafíos relacionados con la falta de interacción activa en muchas herramientas existentes, lo que limita su potencial para promover un aprendizaje autónomo y profundo.

La detección y prevención del plagio emergen como una necesidad crítica en la enseñanza de programación, pero el enfoque predominante en la detección deja un vacío en estrategias preventivas efectivas. Herramientas como MOSS y CodeSight destacan en la identificación de

similitudes en código, pero las oportunidades de innovación incluyen algoritmos más avanzados y estrategias formativas que reduzcan la incidencia de plagio desde etapas tempranas. Incorporar herramientas que promuevan la ética académica, como sistemas de control de versiones y entornos de evaluación controlados, podría complementar los esfuerzos actuales y fomentar un aprendizaje más auténtico.

En cuanto a la retroalimentación automática, las herramientas dinámicas como jueces automáticos y librerías de pruebas unitarias están ampliamente adoptadas, pero enfrentan críticas por la calidad limitada de sus comentarios. La necesidad de retroalimentación más detallada y accesible impulsa el desarrollo de herramientas híbridas y estáticas que no solo evalúen respuestas, sino que también guíen al estudiante en la resolución de errores. Además, la implementación de librerías de pruebas unitarias, aunque menos frecuente en entornos educativos, podría optimizar los recursos disponibles al reducir la infraestructura necesaria, ofreciendo una solución viable para cursos masivos.

Finalmente, los aspectos técnicos y contextuales resaltan el impacto positivo de las herramientas gratuitas y de código abierto, así como de las aplicaciones web integradas en plataformas como Moodle, en la accesibilidad y flexibilidad de los laboratorios virtuales. Sin embargo, la corta vida útil de muchas herramientas educativas y la falta de soporte limitan su aplicabilidad a largo plazo. La sostenibilidad y la expansión del soporte a lenguajes menos representados, como los funcionales, son retos clave que, de abordarse, podrían diversificar y enriquecer las opciones disponibles para docentes y estudiantes en la enseñanza de programación.

4.4 Amenazas a la validez del mapeo de literatura

En el análisis de la validez de esta revisión literaria, identificamos amenazas en distintas categorías.

Amenazas a la validez interna: La selección de publicaciones se realizó a inicios de 2023, lo que implica el riesgo de que ciertos avances académicos recientes, especialmente en el campo de las herramientas de inteligencia artificial, no hayan sido capturados. Este campo ha experimentado una rápida evolución entre 2022 y 2024, lo que podría haber generado nuevas investigaciones, métodos y enfoques que no se reflejan en la literatura analizada. Además, debido a la pandemia de COVID-19, muchas investigaciones se enfocaron en modelos de enseñanza completamente remota. Aunque este tema fue relevante en su momento, la continuidad de esta tendencia es incierta al momento de la publicación de este documento, lo que limita la aplicabilidad de los resultados a un contexto actual que podría diferir del contexto durante el período de revisión. Para mitigar estos posibles sesgos, se adoptaron diversas estrategias. Primero, cada paso del proceso de revisión fue validado por tres individuos de manera independiente, lo que garantizó un análisis más exhaustivo y equilibrado. Además, se siguió un enfoque sistemático, minimizando la influencia de opiniones personales y favoreciendo la objetividad en la inclusión de fuentes. Finalmente, para evitar la omisión de estudios relevantes, se definieron claramente los criterios de inclusión y exclusión, así como la cadena de búsqueda utilizada, lo que aseguró que se capturaran todos los estudios pertinentes dentro del alcance del tema de investigación.

Amenazas a la validez externa: Las circunstancias particulares del periodo de la pandemia influyeron fuertemente en las metodologías y enfoques educativos estudiados, especialmente en las soluciones remotas. Sin embargo, estas pueden no ser tan relevantes en entornos donde se ha

retomado la enseñanza presencial o híbrida. Por lo tanto, la tendencia observada en estos estudios podría no ser representativa de escenarios futuros, donde las condiciones de enseñanza o el papel de la inteligencia artificial (IA) hayan cambiado sustancialmente. Para mitigar esta amenaza, confiamos en que la revisión incluye un número suficiente de publicaciones, abarcando un periodo amplio de tiempo desde 2012. Esto reduce la influencia de las condiciones temporales y asegura que los resultados sean más robustos y representativos a largo plazo.

Amenazas a la validez de constructo: Una amenaza a la validez de constructo en este mapeo es la posible inconsistencia en la definición de conceptos clave como *"automatic feedback"*, *"blended learning"*, *"automatic assessment"* o *"autograder"*, que pueden ser interpretados de manera diferente según el autor o el contexto. Para mitigar esta amenaza, se contó con la asesoría de expertos y se validaron los conceptos utilizados a través de una revisión detallada de la literatura, asegurando que las definiciones fueran consistentes con los marcos teóricos y enfoques más establecidos en el campo de la educación y las tecnologías educativas.

Amenazas a la validez estadística: Una amenaza en este mapeo es el sesgo derivado de la concentración de estudios sobre enseñanza remota durante la pandemia, lo que podría afectar la representatividad de los resultados. Además, el tamaño de la muestra o la diversidad de los estudios incluidos podría influir en los resultados. Para mitigar estas amenazas, se utilizó una estrategia que incluyó varias bases de datos académicas, una cadena de búsqueda bien definida y criterios de inclusión y exclusión claros, lo que permitió obtener una muestra más representativa. También se contó con la validación de expertos y la revisión exhaustiva de la literatura para asegurar la coherencia en la selección de los estudios.

4.5 Conclusiones

En este estudio se analizaron 98 publicaciones, clasificadas en diversas categorías, y se identificaron avances significativos y áreas de mejora en el uso de herramientas para la enseñanza introductoria de programación en ambientes universitarios.

Las herramientas de retroalimentación automática destacan por su aporte educativo y utilidad administrativa, ya que facilitan el trabajo de los docentes mediante evaluaciones rápidas y objetivas, además de proporcionar a los estudiantes retroalimentación oportuna sobre su desempeño. La programación visual se reconoce como una estrategia efectiva para introducir a los estudiantes a la programación. Esto, junto con estrategias creativas, como la gamificación y el uso de temas de interés en los cursos, ha demostrado ser efectivo para captar y mantener el compromiso de los estudiantes. Asimismo, se identificó que las herramientas basadas en la web se posicionan como una solución preferida debido a su accesibilidad y facilidad de implementación en cursos universitarios.

No obstante, persisten retos significativos, especialmente en la sostenibilidad y desarrollo de herramientas, como las de visualización de algoritmos, que enfrentan limitaciones técnicas y operativas. En el caso de la programación visual, la falta de herramientas de retroalimentación automática limita su adopción. Por otra parte, en el ámbito del plagio, las estrategias actuales se centran principalmente en la detección reactiva de similitudes, dejando desatendidas metodologías preventivas que podrían fomentar prácticas más éticas y educativas. Además, el abandono

frecuente de herramientas y su escaso soporte a largo plazo, incluso en aquellas de código abierto, restringe su aplicación en entornos educativos más amplios.

A pesar de estos desafíos, existe una oportunidad significativa para desarrollar herramientas integradas que soporten diversos casos de uso, como la programación visual, la implicación y la retroalimentación automática, mejorando tanto el aprendizaje de los estudiantes como la eficiencia de los docentes. También es necesario explorar estrategias preventivas más efectivas en la detección del plagio y fortalecer el compromiso de los estudiantes mediante enfoques innovadores. Un enfoque equilibrado en estas áreas, junto con una inversión en herramientas educativas más sostenibles en el tiempo, podría transformar la enseñanza de la programación, asegurando un impacto duradero y un aprendizaje más significativo.

Capítulo 5: Propuesta de marco de implementación de laboratorios de programación

Como se observa en capítulos anteriores, existe una gran variedad de herramientas, lenguajes de programación, casos de uso y contextos en los cuales puede ser necesaria una herramienta para la enseñanza de la programación introductoria. Sin embargo, no encontramos en la literatura una propuesta de marco o metodología que permita definir o seleccionar una herramienta específica según las necesidades particulares de cada contexto educativo.

Por lo tanto, en este capítulo proponemos una aplicación web llamada **Tool-Finder**, diseñada para facilitar la búsqueda de las herramientas que mejor se ajusten a los objetivos y contenidos del curso de programación. Esta herramienta emplea un sistema de filtros basado en elementos de contexto relevantes, identificados a partir de la investigación realizada, para ayudar a los docentes a seleccionar la opción más adecuada según los requerimientos de su curso.

La Figura *18* ilustra el proceso que se siguió para la creación de la aplicación. Este se organizó en cuatro fases principales: (i) Identificación de los elementos de contexto (sección 5.1), fase en la que se analizaron los factores clave para seleccionar herramientas educativas en programación; (ii) Diseño de la aplicación (sección 5.2), etapa en la que se definieron la estructura, interfaz y funcionalidad básica del sistema; (iii) Construcción de la aplicación (sección 5.3), donde se implementaron los componentes técnicos y se integraron los filtros de búsqueda; y (iv) Demostración (sección 5.4), fase destinada a evaluar y mostrar el funcionamiento de la aplicación en escenarios de prueba, validando su utilidad para docentes en contextos de enseñanza de programación.

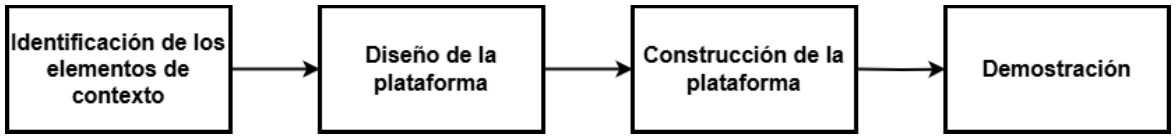

Figura 18. Proceso de desarrollo de la propuesta

5.1 Identificación de los elementos de contexto

En la enseñanza de programación introductoria, existen múltiples contextos que influyen en la selección de herramientas, ya que cada curso y entorno educativo presenta necesidades y limitaciones específicas. Por ejemplo, en algunos casos, el costo de la herramienta es un factor crucial, ya que las instituciones pueden optar por herramientas gratuitas o de código abierto para reducir gastos. En otros contextos, la plataforma puede ser determinante, ya que aplicaciones web suelen preferirse para evitar la instalación en dispositivos de los estudiantes, mientras que algunas herramientas requieren un servidor propio o el uso de servicios de terceros, lo cual puede complicar su implementación. También es esencial considerar el lenguaje de programación soportado por la herramienta, ya que debe coincidir con el lenguaje del curso, y el nivel de soporte y documentación disponible, que asegura su estabilidad y actualizaciones a lo largo del tiempo.

En esta sección exploramos cómo los diversos elementos de contexto influyen en la selección de herramientas para cursos de programación, permitiendo una alineación más precisa con los objetivos y necesidades de cada curso. Con base en el mapeo de literatura realizado, analizamos la relevancia de estos elementos en el aprendizaje de programación y describimos cada uno de los

factores clave identificados, junto con los parámetros definidos para su evaluación en la herramienta Tool-Finder.

La Tabla 14 presenta un desglose superficial de los elementos de contexto considerados en la aplicación, que serán definidos más a profundidad en esta sección.

Elemento de contexto	**Descripción**	**Opciones**
Costo de la herramienta	Precio o tarifa para el uso de la herramienta, considerando licencias y suscripciones.	Sin costo, pago, prueba gratuita
Plataforma	Entorno en el que se utiliza la herramienta, evaluando accesibilidad e instalación.	Web, nativa
Servidor o *hosting*	Sistema de alojamiento necesario para ejecutar la herramienta.	De terceros, propio
Lenguaje de programación soportado	Lenguajes de programación que la herramienta apoya en su funcionalidad y operaciones.	Python, Java, C/C++, Scratch, entre otros
Soporte y documentación	Respaldo y asistencia proporcionados para la herramienta, incluyendo actualizaciones.	Soportada, no soportada, documentada, no documentada
Extensibilidad	Capacidad de la herramienta para ampliarse o adaptarse a nuevas funcionalidades.	Extensible, no extensible
Interés o enfoque principal	Intención principal de la herramienta y problemas específicos que aborda en programación, basados en los casos de uso definidos en la sección 4.2.4.	Programación visual, plagio, visualización de algoritmos, retroalimentación automática, asistencia a estudiantes, implicación
Características adicionales	Funcionalidades adicionales que amplían el potencial de la herramienta en entornos educativos.	Colaboración, accesibilidad, detección de plagio, personalización, etc.

Tabla 14. Elementos de contexto considerados para la aplicación

Costo de la herramienta

El "costo de la herramienta" se refiere a los gastos asociados con su uso en el proceso de aprendizaje, incluyendo licencias de software, suscripciones a servicios en línea, cursos pagos o cualquier otro recurso que los estudiantes o las instituciones necesiten adquirir para acceder a la herramienta.

Evaluar el costo es esencial para determinar si los beneficios de la herramienta justifican el gasto, considerando que algunas son gratuitas, de código abierto o ofrecen versiones de prueba gratuitas, mientras que otras requieren pagos regulares o únicos.

Para el contexto del costo, hemos definido tres parámetros: (i) sin costo, para herramientas que no requieren pago alguno para su uso, instalación o *hosting*; (ii) de pago, para aquellas que implican algún costo para su uso o adquisición; y (iii) prueba gratuita, para herramientas de pago que ofrecen una opción de uso gratuito temporal o limitada.

Plataforma

La "plataforma" de la herramienta se refiere al entorno en el que los usuarios pueden utilizarla, evaluando si se requiere o no una instalación. Este elemento de contexto permite analizar la accesibilidad de la herramienta y la facilidad con la que los estudiantes pueden acceder a ella.

Al elegir una herramienta para un curso de programación, es crucial considerar su accesibilidad y si el entorno será cómodo para los usuarios. En general, las aplicaciones web suelen preferirse frente a aquellas que requieren instalación, ya que facilitan el acceso y el uso sin configuraciones adicionales.

Para este elemento de contexto, hemos definido dos parámetros posibles para la plataforma: (i) web, cuando la herramienta se usa directamente en línea, y (ii) nativa, cuando requiere instalación en un dispositivo.

Servidor o hosting

Un servidor o *"hosting"* es un sistema informático que proporciona los recursos necesarios para alojar y ejecutar aplicaciones o sitios web. En algunos casos, las aplicaciones requieren la instalación de un servidor propio, especialmente cuando no hay un proveedor tercero que ofrezca el servicio deseado de manera gratuita o de pago. Esto implica la necesidad de que la organización o el docente configure y mantenga su propio servidor físico o virtual, lo cual representa una inversión adicional en hardware, software y administración del sistema.

Al evaluar una herramienta para un curso de programación, es crucial considerar si requiere un servidor propio o la adquisición de un servicio de terceros, ya que esto impacta en los costos y el mantenimiento. Estos factores pueden ser limitantes en ciertos escenarios, sobre todo en entornos educativos con recursos limitados.

Para el servidor de la herramienta, hemos definido los siguientes parámetros posibles: (i) de terceros, cuando la herramienta depende de un proveedor externo, y (ii) propio, cuando es necesario implementar un servidor propio.

Lenguaje de programación soportado

El "lenguaje de programación soportado" por una herramienta de apoyo en el aprendizaje de programación se refiere a los lenguajes específicos que la herramienta está diseñada para enseñar, facilitar y respaldar durante el proceso de aprendizaje. Estas herramientas suelen ofrecer funciones como resaltado de sintaxis, autocompletado, depuración, ejecución de código y detección de errores para uno o varios lenguajes populares y ampliamente utilizados.

Al evaluar una herramienta para un curso de programación, es fundamental considerar los lenguajes que soporta, ya que esto determina su capacidad para alinearse con los objetivos de aprendizaje y los contenidos del curso. Por ejemplo, no sería adecuado elegir una herramienta que solo soporte Java si el curso está centrado en Python.

Entre los lenguajes soportados se incluye una variedad de opciones tanto convencionales como de programación visual, tales como Python, Java, C, C++, Scratch, entre otros.

Soporte y documentación

El "soporte y documentación" de una herramienta hace referencia al respaldo que el desarrollador o proveedor ofrece, incluyendo actualizaciones, corrección de errores, documentación, atención al cliente y asistencia técnica continua. Una herramienta soportada cuenta con recursos dedicados que garantizan su funcionalidad, seguridad y compatibilidad con nuevos sistemas. En cambio, una herramienta sin soporte oficial depende totalmente del usuario para su mantenimiento, lo que puede limitar su uso en entornos formales.

Al seleccionar una herramienta para un curso de programación, es fundamental considerar si cuenta con soporte y documentación adecuados, ya que la falta de estos puede llevar a problemas a largo plazo. Sin un respaldo adecuado, una herramienta corre el riesgo de volverse obsoleta o incompatible con el tiempo, comprometiendo su viabilidad en entornos educativos o profesionales.

Para evaluar el soporte, hemos definido los parámetros: (i) soportada, cuando ha recibido actualizaciones en el último año, y (ii) no soportada, cuando carece de ellas. En cuanto a la documentación, los parámetros definidos son: (i) documentada, y (ii) no documentada.

Extensibilidad

La "extensibilidad" de una herramienta se refiere a su capacidad para ser ampliada, mejorada o adaptada a nuevas funcionalidades o requisitos más allá de su propósito inicial. Una herramienta extensible está diseñada de manera modular y flexible, lo que permite a desarrolladores o usuarios agregar, modificar o reemplazar componentes sin alterar todo el sistema. Esto facilita que la herramienta se mantenga actualizada y relevante conforme surgen nuevas necesidades.

Al evaluar una herramienta para un curso de programación, es importante considerar si puede extenderse y si existen extensiones públicas disponibles. Una herramienta extensible permite adaptarse a las necesidades cambiantes de los cursos sin necesidad de reemplazar la solución existente, lo cual resulta particularmente útil en entornos educativos.

Para la extensibilidad de la herramienta, hemos definido dos parámetros: (i) extensible, cuando permite modificaciones o mejoras adicionales, y (ii) no extensible, cuando no ofrece esta flexibilidad.

Interés o enfoque principal

El "interés o enfoque principal de una herramienta" está relacionado con la categorización establecida en el mapeo sistemático de literatura, lo que permite identificar la intención principal de la herramienta y el problema específico que mejor resuelve en la enseñanza de programación. Estos enfoques fueron definidos con base en un conjunto de casos de uso frecuentes en cursos introductorios, abordando las distintas necesidades pedagógicas y técnicas observadas en los laboratorios virtuales.

Al evaluar una herramienta para un curso de programación, es importante considerar el propósito específico para el que se necesita y si su funcionalidad se ajusta a las necesidades del curso. Esto implica analizar si la herramienta aportará valor en áreas clave del aprendizaje, alineándose con los objetivos pedagógicos y facilitando el proceso educativo según las demandas del contexto educativo.

Para el interés o enfoque principal de cada herramienta, hemos establecido los siguientes parámetros: (i) programación visual, utilizada para reducir la barrera de la sintaxis y facilitar el aprendizaje de conceptos básicos; (ii) plagio, para identificar y prevenir la copia de código entre estudiantes; (iii) visualización de algoritmos, que ayuda a los estudiantes a comprender mejor la ejecución de un programa y la estructura de datos; (iv) retroalimentación automática, que permite proporcionar retroalimentación en tiempo real, aliviando la carga laboral de los docentes; (v) asistencia a estudiantes, que apoya en la resolución de errores y la comprensión de conceptos difíciles; y (vi) implicación, que busca motivar a los estudiantes a participar activamente y a mantener el interés en el curso.

Características adicionales

Las características adicionales de una herramienta de desarrollo abarcan funcionalidades que complementan los elementos de contexto principales, ampliando su utilidad en el aprendizaje de programación. Estas pueden incluir capacidades de depuración y pruebas, herramientas de colaboración (como seguimiento de cambios y revisión de código), opciones de accesibilidad, detección de plagio, programación visual, visualización de algoritmos y características que fomentan la implicación y motivación de los estudiantes. Estas funcionalidades permiten adaptar la herramienta a los objetivos específicos del curso y las necesidades de los estudiantes, incrementando su valor en el proceso educativo.

5.2 Diseño de la aplicación

El diseño de la aplicación se desarrolló en dos etapas clave. En la primera etapa, se elaboró un mock o prototipo inicial como representación visual de la aplicación, lo que permitió analizar y definir las funcionalidades que debía cumplir la aplicación. En la segunda etapa, se diseñó una arquitectura guía que estableció los componentes principales de la aplicación y sus relaciones.

La Figura 19 muestra el mock de la página principal de la aplicación, en el que se distinguen los siguientes componentes principales: (i) una barra de navegación, (ii) filtros para seleccionar las

aplicaciones de acuerdo a los elementos de contexto definidos en la sección 5.1, (iii) una barra de filtros activos que permite desactivarlos rápidamente para explorar las herramientas de forma ágil y (iv) una lista de las herramientas que cumplen con los filtros activos, junto a una breve descripción e información adicional. Cabe destacar que el diseño propuesto busca sugerir herramientas basándose en los elementos de contexto, por lo que se excluye la posibilidad de realizar búsquedas textuales de las aplicaciones, alineándose con el objetivo de una exploración de acuerdo con las necesidades del usuario.

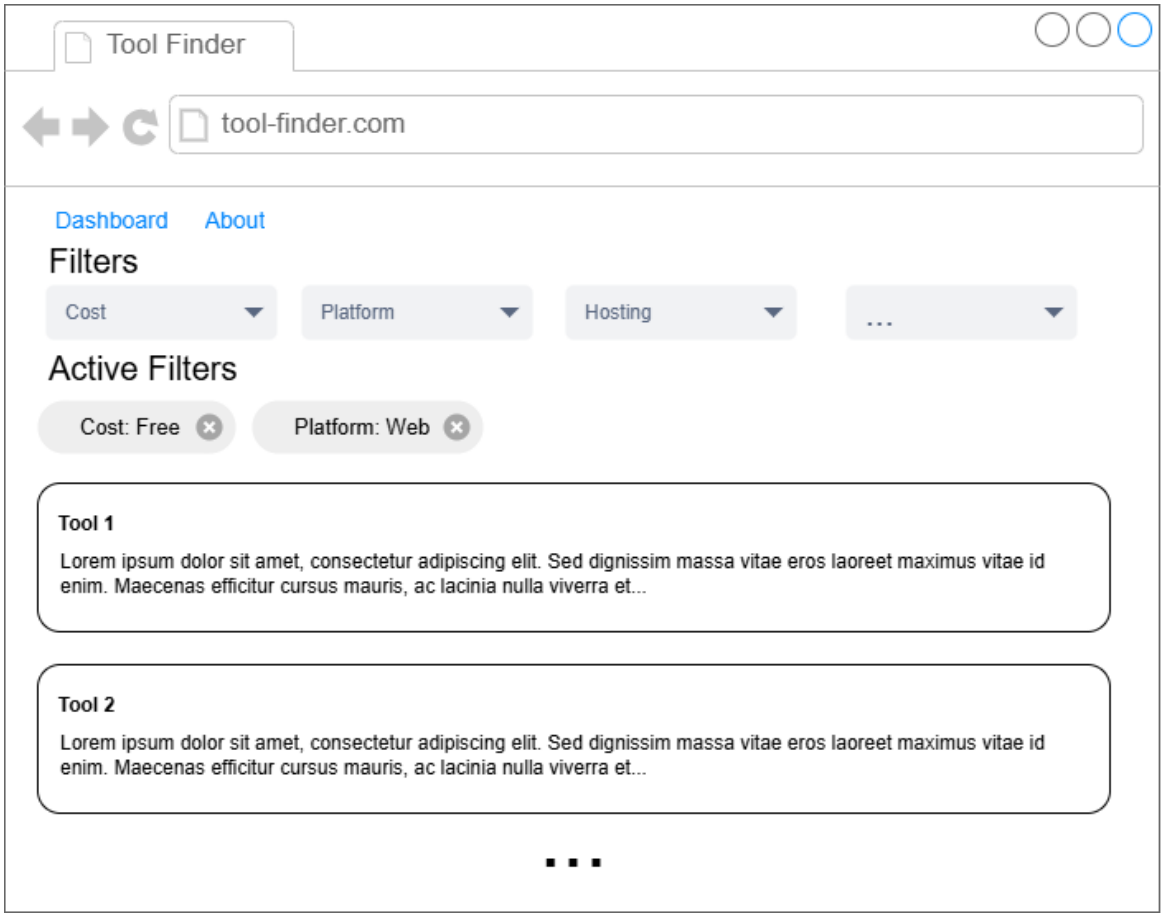

Figura 19. Mock de la aplicación

Una vez definidas las funcionalidades básicas de la aplicación, se procedió a diseñar una arquitectura sencilla acorde con los principios de un MVP (Producto Mínimo Viable). La Figura 20 ilustra este diseño básico, que implementa una arquitectura cliente-servidor para una aplicación web. El diseño se estructuró separando las vistas de los modelos de datos para simplificar su desarrollo y mantenimiento. En cuanto a las vistas, se incluyen dos páginas principales: la página de *"dashboard"* o "principal", donde los usuarios pueden filtrar y visualizar las herramientas de interés, y la página de *"about us"* o "acerca de", destinada a proporcionar información sobre la aplicación. Los modelos, por su parte, abarcan los elementos esenciales del sistema, que son los filtros y las herramientas. Además, se cuenta con una fuente de datos que almacena las herramientas y sirve como base para las funcionalidades principales de la aplicación.

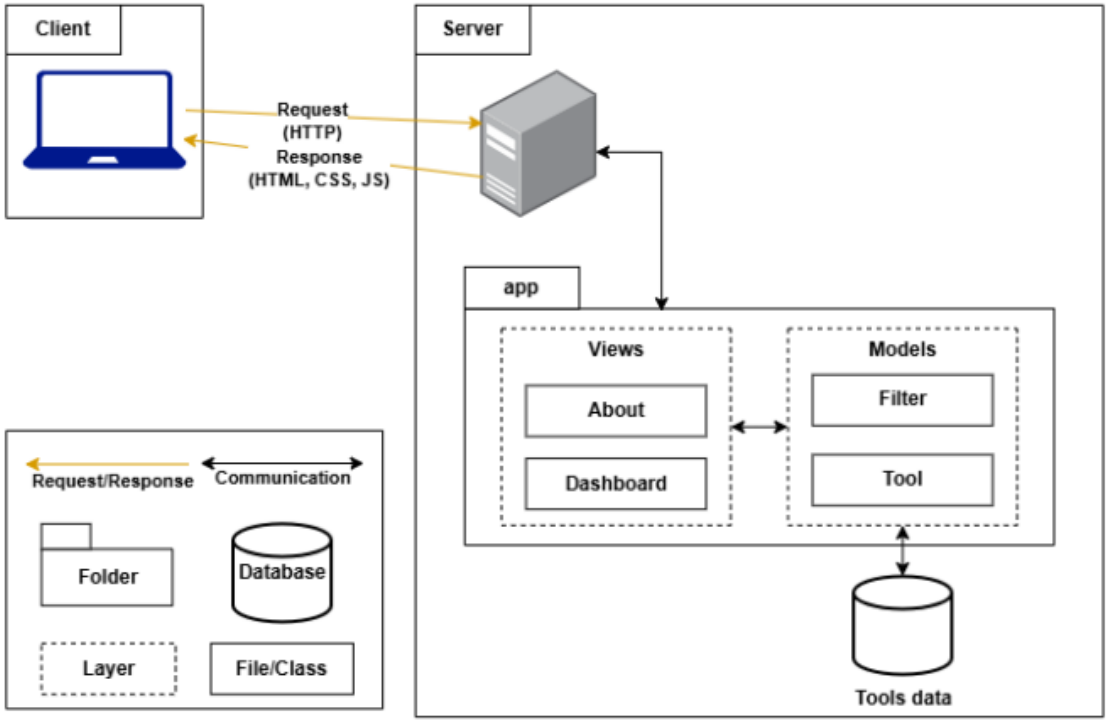

Figura 20. Arquitectura de la aplicación

5.3 Construcción de la aplicación

Tomando como base el diseño anterior, la aplicación fue desarrollada utilizando el lenguaje de programación JavaScript en combinación con el framework NextJS. Las principales razones para seleccionar NextJS fueron su amplio reconocimiento en la industria, su excelente documentación y su facilidad para el despliegue, lo que permitió crear un producto mínimo viable (MVP) en un tiempo razonable.

La Figura *21* presenta la arquitectura final sobre la cual se construyó la aplicación. En esta se detalla el uso de los navegadores Mozilla Firefox y Google Chrome, en los que se verificó el correcto funcionamiento de la aplicación. Además, se especifica el empleo de NextJS como framework principal, junto con la incorporación de una capa de “Componentes”, encargada de encapsular los elementos React relevantes para la aplicación. También se indica el sistema implementado como fuente de datos.

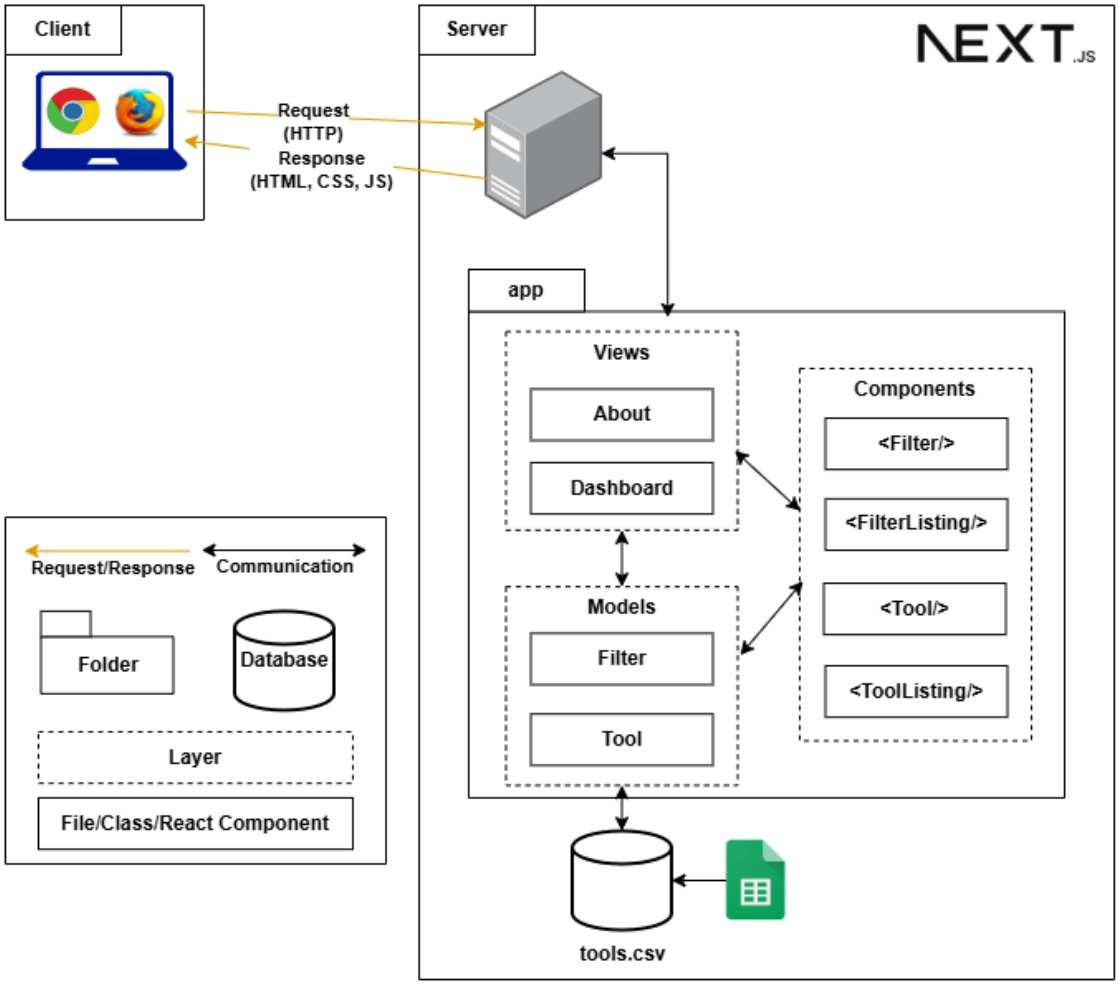

Figura 21. Arquitectura final de la aplicación

La aplicación utiliza un archivo CSV como fuente de datos, descargado directamente desde Google Sheets. Este archivo se actualiza manualmente en el código fuente de la aplicación cuando se producen cambios en los datos. Este enfoque permitió contar con una fuente de datos fácilmente editable, evitando la necesidad de implementar un sistema de administración de información (CRUD), ya que no era un objetivo principal de esta investigación. Además, esta decisión ayudó a reducir significativamente el tiempo de desarrollo y la complejidad asociada con el despliegue. Sin embargo, la aplicación se diseñó con la flexibilidad suficiente para integrar una base de datos más robusta o fuentes de datos alternativas en el futuro, si fuese necesario. La Figura 22 muestra un ejemplo del archivo fuente de datos alojado en Google Sheets, disponible públicamente en https://bit.ly/3CBxPU6 y abierto a comentarios y sugerencias por parte del público.

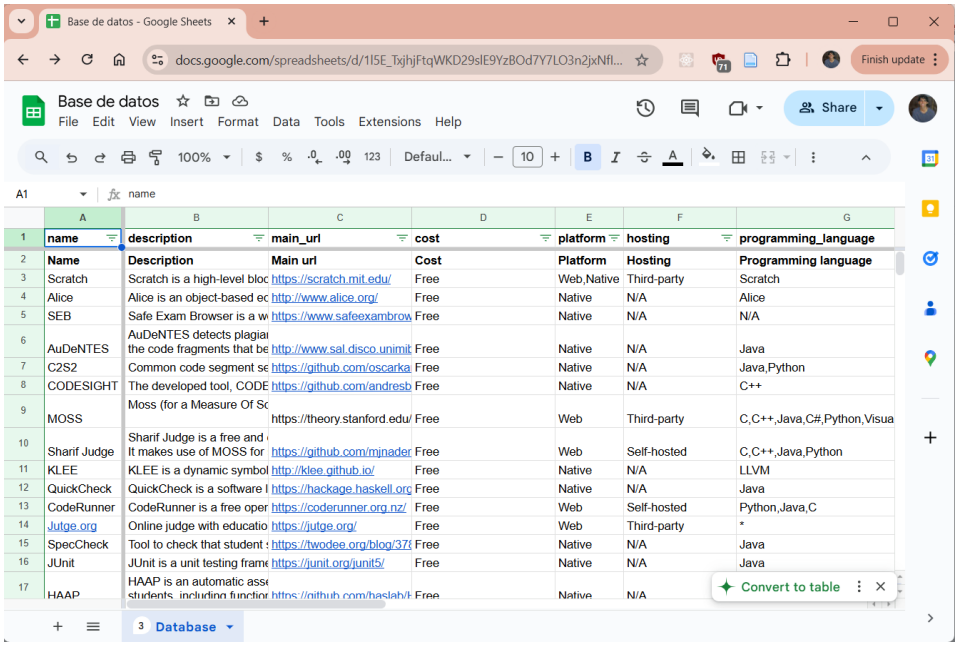

Figura 22. Fuente de datos alojada en Google Sheets

Durante el proceso de implementación de la aplicación, se utilizó el entorno de desarrollo integrado (IDE) WebStorm, conocido por sus herramientas avanzadas para el desarrollo web y su soporte nativo para tecnologías como React y NextJS. La Figura 23 muestra el desarrollo de la página de *dashboard* de la aplicación y la Figura 24 presenta el desarrollo del componente *"ToolListing"*, encargado de listar las herramientas en el *dashboard*, tal como su nombre lo indica. Ambos son componentes fundamentales de la aplicación.

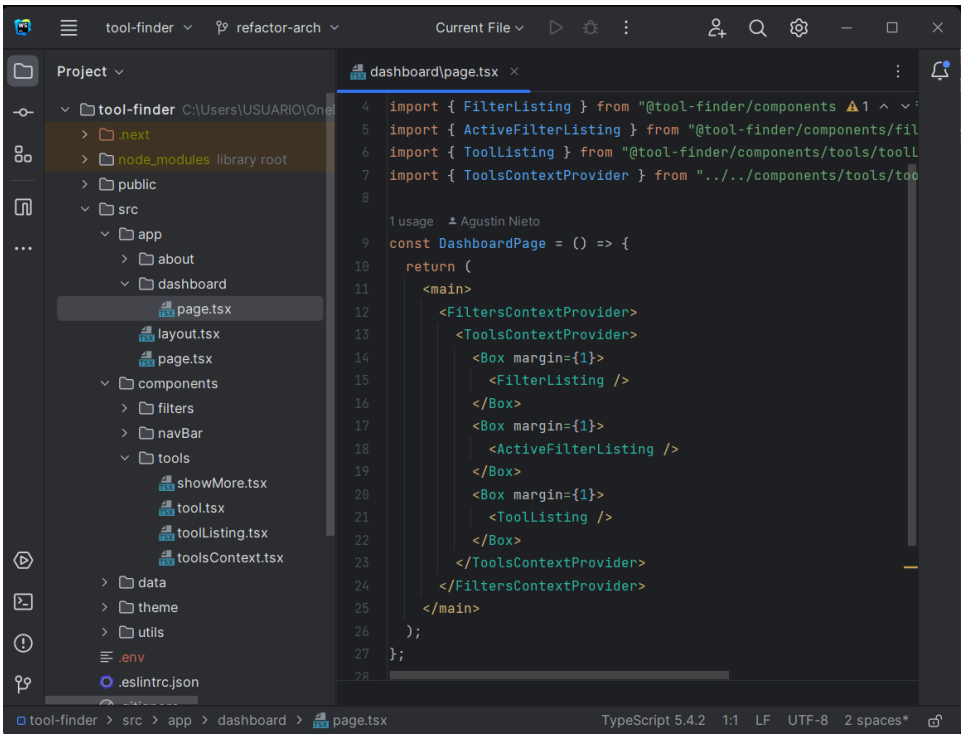

Figura 23. Demostración del desarrollo del *dashboard* de la aplicación en Webstorm

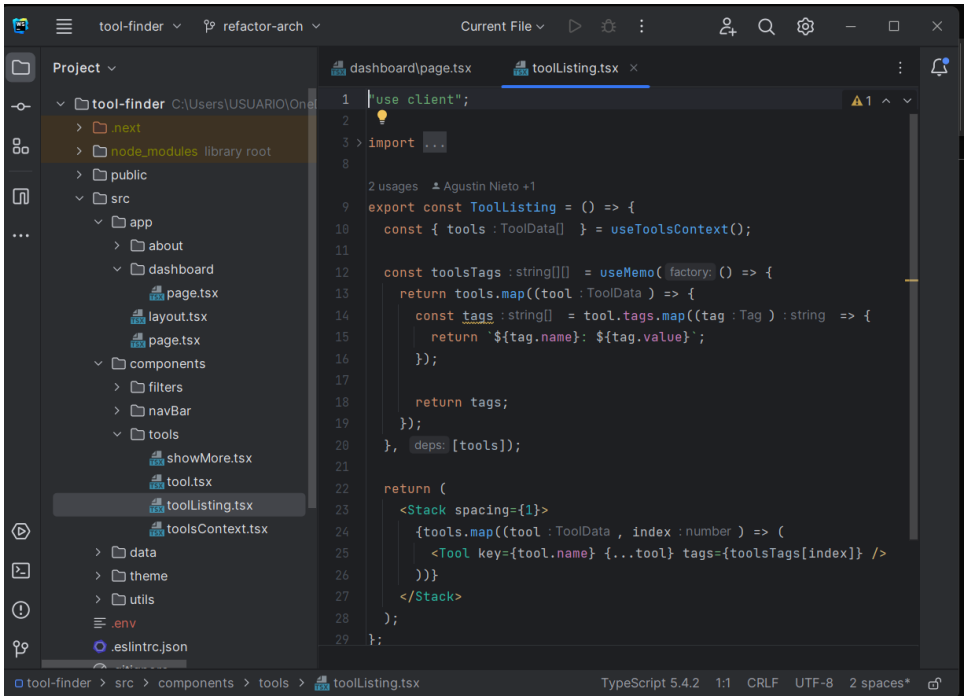

Figura 24. Demostración del desarrollo de un componente en Webstorm

Para la gestión de versiones se empleó GitHub como repositorio principal. La Figura *25* presenta una captura de pantalla del repositorio donde se aloja el código fuente de la aplicación. Asimismo, se utilizó Trello para el seguimiento de tareas y la coordinación del desarrollo.

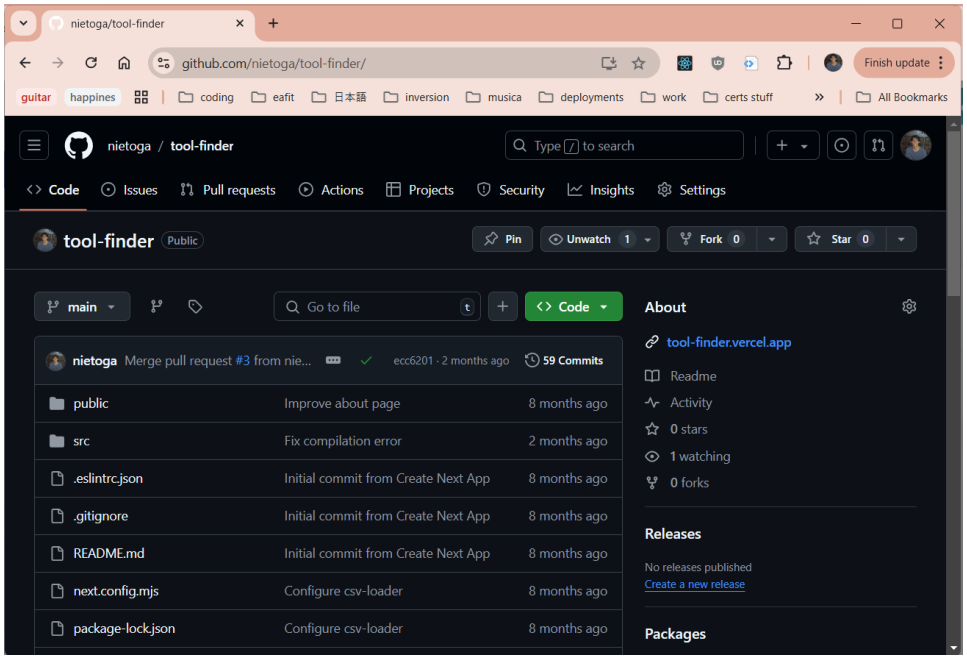

Figura 25. Github de la aplicación

El despliegue de la aplicación se realiza automáticamente mediante Vercel, aprovechando su integración nativa con NextJS. En la Figura *26* se muestra el panel de control de Vercel, utilizado para la administración de los despliegues.

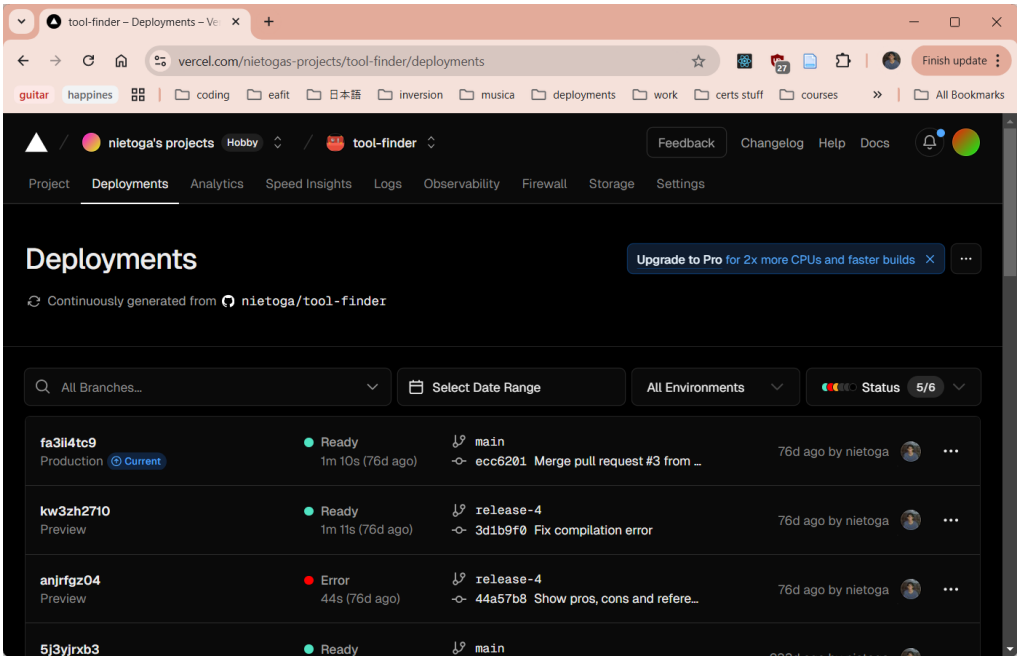

Figura 26. Panel de despliegues de la aplicación, en Vercel

El código fuente de la aplicación está disponible públicamente en el siguiente enlace: https://github.com/nietoga/tool-finder

5.4 Demostración

La aplicación está disponible en https://tool-finder.vercel.app/ y cuenta con un "*dashboard*" o "página principal" que incluye un sistema de filtrado para ajustar la visualización de herramientas según los criterios seleccionados. Además, cuenta con una página *"about us"* o "acerca de", diseñada únicamente con fines informativos, donde los usuarios pueden conocer el propósito de la herramienta y dejar sugerencias o explorar la fuente de datos.

Dashboard

El *dashboard* es la página principal de la aplicación, donde se listan las aplicaciones disponibles y que cuenta con filtros para cada elemento de contexto, permitiendo ajustar la visualización según los criterios seleccionados. Esta página siempre mostrará únicamente las herramientas que cumplen con los filtros aplicados.

La Figura 27 presenta el diseño del *dashboard* de la aplicación, en el cual se observan los filtros ubicados al inicio de la página, seguidos de la lista de herramientas que cumplen con los criterios seleccionados. Cada herramienta está representada con su nombre y elementos de contexto en la parte superior, mostrados como chips, además de un enlace a su página principal y una breve descripción. En la parte inferior de cada tarjeta de herramienta, un botón de "mostrar más" permite desplegar información adicional, como publicaciones de referencia, enlaces a su repositorio o documentación y detalles sobre características adicionales, proporcionando así una descripción completa de cada herramienta.

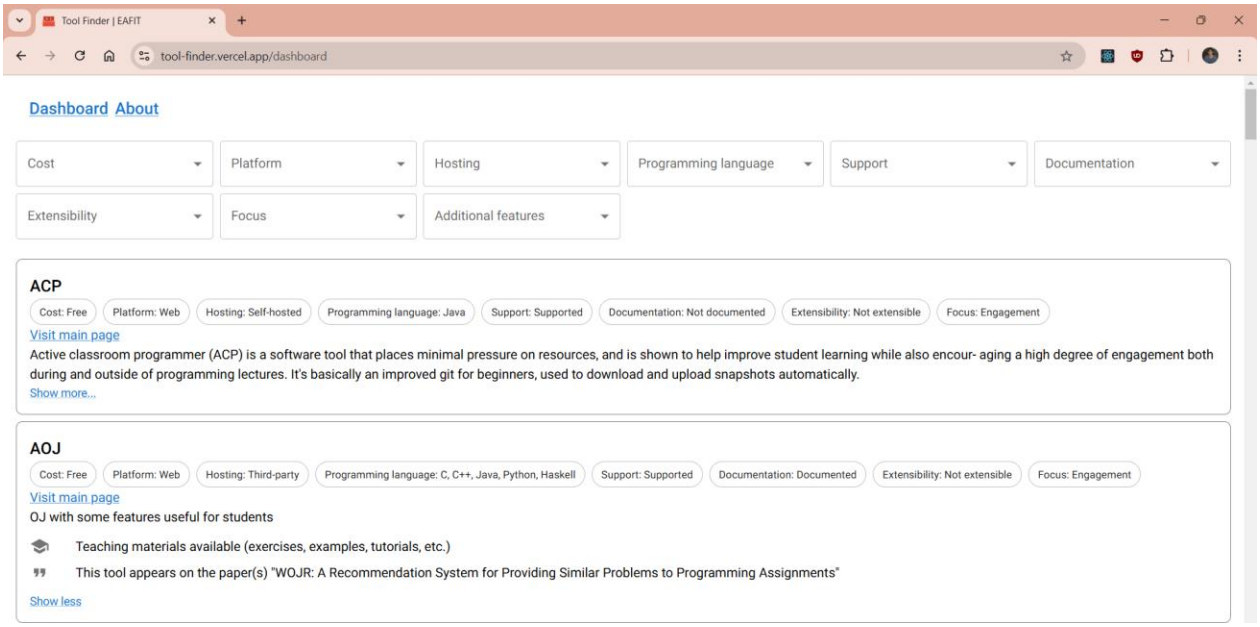

Figura 27. Vista general del *dashboard* de Tool-Finder

Página "about us"

La página *about us* invita a los usuarios a dejar sugerencias o explorar la base de datos de la aplicación. La Figura 28 muestra esta página, la cual es meramente informativa y proporciona detalles sobre el propósito de la herramienta, orientando a los usuarios en cómo pueden contribuir o aprovechar al máximo sus funcionalidades.

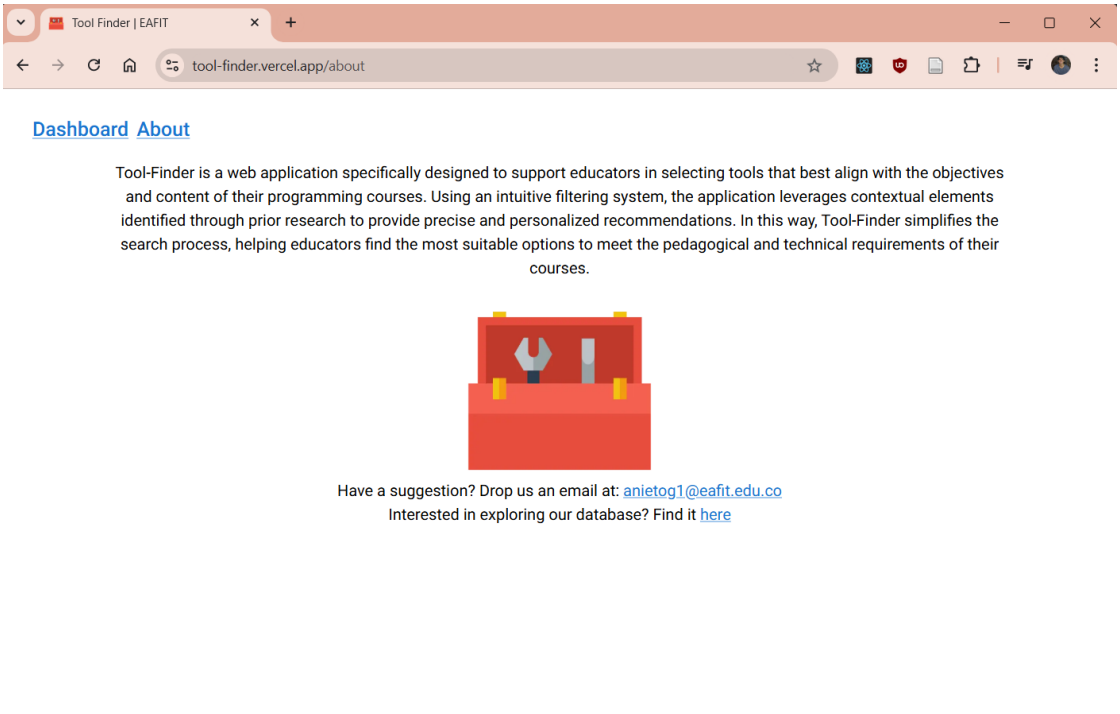

Figura 28. Página *about us* de la herramienta

Sistema de filtrado

El sistema de filtrado de la aplicación consta de dos partes: (i) los filtros, basados en los elementos de contexto, y (ii) los chips que aparecen debajo de los filtros y permiten eliminar un filtro previamente configurado. Los filtros permiten reducir el conjunto de aplicaciones disponibles según el contexto y necesidades específicas de un curso o situación de aprendizaje, facilitando la selección de herramientas relevantes.

La Figura 29 muestra un ejemplo de filtrado que busca una aplicación web, gratuita, para la visualización de algoritmos. El sistema responde adecuadamente, sugiriendo PythonTutor como la única alternativa que cumple con estas características en nuestra base de datos. PythonTutor es una herramienta interactiva en línea que permite a los estudiantes visualizar paso a paso la ejecución de código en varios lenguajes, incluido Python, facilitando la comprensión de estructuras de datos y el flujo de control en algoritmos.

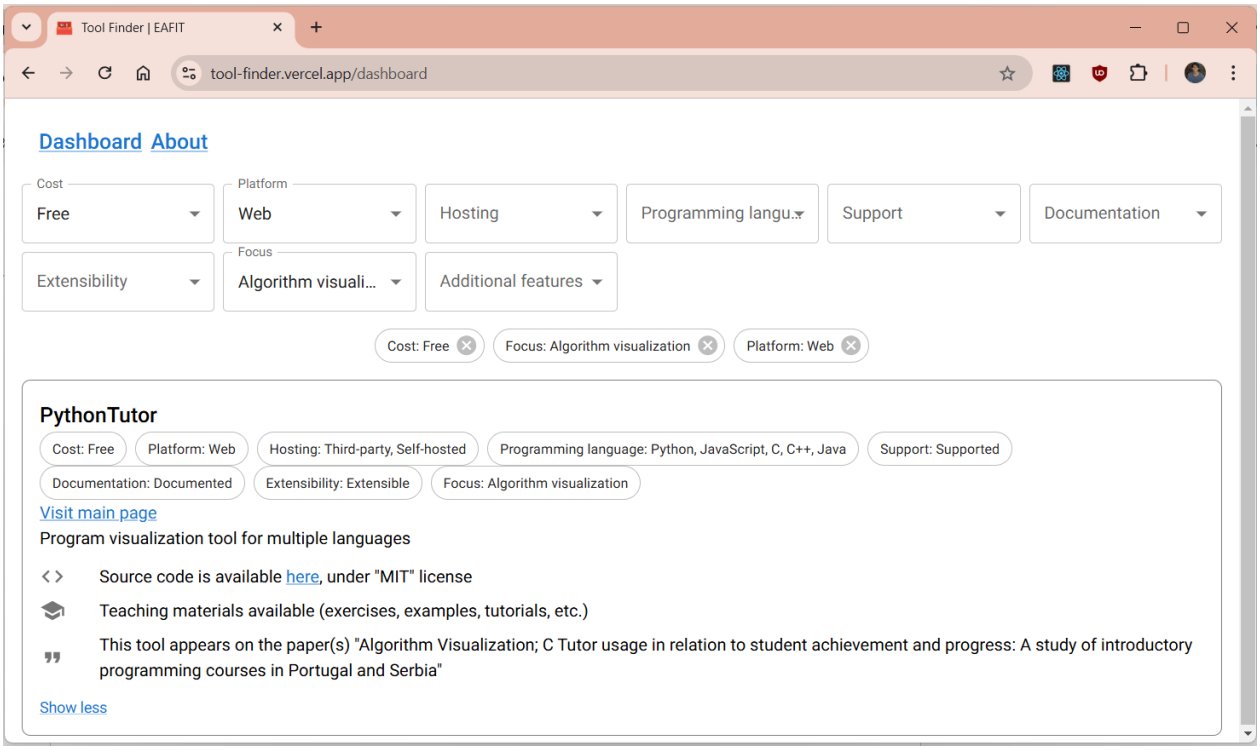

Figura 29. Filtrado de una aplicación web, gratuita, para la visualización de algoritmos

La *Figura 30* muestra el resultado de un filtrado para encontrar herramientas de retroalimentación automática que soporten Python, C o C++ y sean aplicaciones web, donde la herramienta responde correctamente ofreciendo varias opciones adecuadas: CodeLab, una plataforma en línea para práctica y retroalimentación automática en múltiples lenguajes de programación; CodeRunner, un plugin de Moodle que permite realizar evaluaciones automáticas con retroalimentación inmediata en Python y otros lenguajes; Jutge.org, un juez automático en línea para la evaluación de problemas de programación; Questournament, una plataforma de concursos de programación en línea que promueve el aprendizaje mediante desafíos; UVa Online Judge, un entorno para la práctica de programación competitiva y académica, y VPL Plugin, otro plugin de Moodle que facilita la configuración de laboratorios virtuales con evaluación automática de código. Estas herramientas

se ajustan al contexto solicitado, proporcionando opciones para retroalimentación automática en Python.

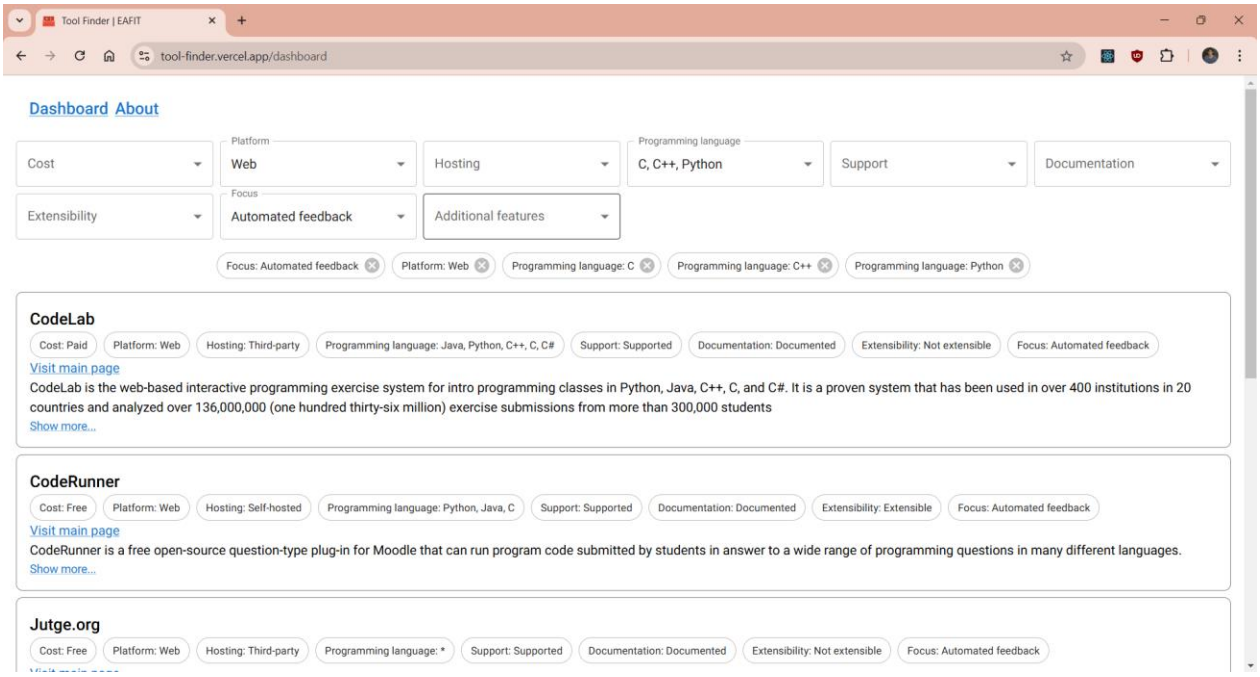

Figura 30. Filtrado de una herramienta web para retroalimentación automática en Python

5.5 Trabajo futuro

Nuestra aplicación, en su fase inicial como Producto Mínimo Viable (MVP), se enfoca en la simplicidad y en cumplir los objetivos de filtrado y clasificación de herramientas educativas. Su diseño actual, establece una base para futuras versiones que podrán incorporar mejoras que aumenten su utilidad y atractivo en el ámbito educativo.

En el futuro, la herramienta podría ampliarse para incluir mejoras que optimicen su funcionalidad y capacidad de selección, como la integración de un sistema de base de datos más robusto que permita gestionar un mayor volumen de herramientas y datos de manera más eficiente. También se podría incorporar un sistema de sugerencias avanzado que, mediante inteligencia artificial a través de servicios como ChatGPT, ofrezca recomendaciones personalizadas según los elementos de contexto definidos. Además, la traducción de la interfaz a distintos idiomas mejoraría su accesibilidad a usuarios internacionales. Otra posible mejora es la implementación de un sistema de validación por terceros, lo cual brindaría mayor seguridad y confiabilidad en la selección de herramientas. Estas ampliaciones permitirían a los usuarios acceder a recomendaciones mejor fundamentadas y a una base de datos más extensa, optimizando el proceso de búsqueda y filtrado en el contexto educativo.

Capítulo 6: Demostración del marco de implementación de laboratorios de programación

Este capítulo presenta el uso de Tool-Finder para seleccionar una herramienta destinada a un curso masivo de programación en la Universidad EAFIT, así como los resultados obtenidos de dicha implementación. La estructura del capítulo se organiza de la siguiente manera: (i) características del curso (sección 6.1), (ii) selección de las herramientas (sección 6.2), (iii) metodología aplicada en el curso (sección 6.3) y (iv) discusión (sección 6.4).

6.1 Características del curso

Para desarrollar una metodología y seleccionar herramientas adecuadas, es fundamental comprender las características específicas del curso en cuestión. En este caso, estudio se centra en un curso masivo de introducción a la programación llamado "Pensamiento Computacional" (PC), ofrecido por la Universidad EAFIT y dirigido a estudiantes de múltiples carreras (ciencias, literatura, psicología, derecho, algunas ingenierías, entre otros). Este curso se imparte cada semestre con una matrícula que supera los 500 estudiantes.

A continuación, se detallan las características principales del curso, organizadas en las categorías (i) datos específicos del semestre en estudio, (ii) formato de las clases y (iii) temáticas del curso.

Datos específicos del semestre en estudio

- **Fecha de inicio:** 23 de enero
- **Fecha de cierre:** 15 de mayo de 2023
- **Cantidad de grupos:** 35
- **Cantidad de docentes:** 21
- **Cantidad de monitores:** 24
- **Cantidad de estudiantes:** 792

Formato de las clases

- **Duración:** Bloque de 3 horas
- **Cantidad de clases:** 16 (1 clase por semana)
- **Espacio:** Aulas de cómputo, con un computador asignado por estudiante

Temáticas del curso

- Algoritmia y pensamiento computacional
- Variables y tipos de datos
- Entrada y salida
- Condicionales simples y múltiples
- Ciclos *while* y *for*
- Estructuras de datos básicas: textos, listas y diccionarios
- Funciones
- Manipulación de archivos
- Librerías

6.2 Selección de las herramientas

Tool-Finder se utilizó para seleccionar las herramientas más adecuadas para este curso. Debido a la escala y al perfil de los participantes, la selección de herramientas accesibles, intuitivas y capaces de proporcionar retroalimentación automatizada es esencial para garantizar una gestión eficiente del curso y un aprendizaje efectivo para los estudiantes.

Para lograr este objetivo, identificamos la necesidad de contar con dos aplicaciones complementarias: una destinada a asistir durante las clases, proporcionando un entorno de práctica inmediato e intuitivo, y otra diseñada para facilitar a los estudiantes la realización de prácticas y ejercicios fuera del aula, promoviendo el aprendizaje autónomo.

Para seleccionar la aplicación que facilitaría las prácticas durante las clases, identificamos 5 necesidades clave: (i) que fuera una aplicación web, para evitar problemas de instalación y permitir a los estudiantes practicar desde casa sin complicaciones, (ii) que asistiera a los estudiantes durante la programación, ofreciendo un entorno interactivo y amigable, (iii) que fuera gratuita, (iv) que ofreciera soporte para Python, el lenguaje base del curso, y (v) que tuviera soporte y documentación adecuados, para que el equipo docente pueda resolver los problemas que surjan de manera eficiente.

La Figura 31 presenta las herramientas recomendadas tras aplicar estos criterios. La aplicación identificó dos opciones viables: (i) Jupyter, una plataforma interactiva que permite crear y compartir documentos que combinan código, visualizaciones y texto explicativo, ampliamente utilizada en educación y análisis de datos, y (ii) Replit, una herramienta en línea para escribir, ejecutar y colaborar en proyectos de programación en múltiples lenguajes, ideal para entornos educativos y colaborativos. Finalmente, se optó por Jupyter, dado que su capacidad para integrar código y explicaciones en un único entorno lo convierte en una herramienta eficaz para el aprendizaje interactivo y autónomo de los estudiantes. Sin embargo, reconocemos que Replit, con su enfoque colaborativo y soporte para una amplia gama de lenguajes, podría ser más adecuado para cursos o proyectos de programación más avanzados.

En la práctica, Jupyter se utilizó a través de Google Colab, “un servicio de Jupyter Notebook que no requiere configuración previa y proporciona acceso gratuito a recursos de cómputo como GPUs y TPUs” [100]. Google Colab es especialmente adecuado para investigaciones en aprendizaje automático, ciencia de datos y educación, ofreciendo un entorno interactivo y accesible sin necesidad de instalación local. Su integración con Google Drive permite a los estudiantes almacenar y acceder a sus apuntes y prácticas desde cualquier lugar, además de compartir archivos de forma sencilla, fomentando la colaboración y asegurando la continuidad del aprendizaje de manera eficiente y práctica.

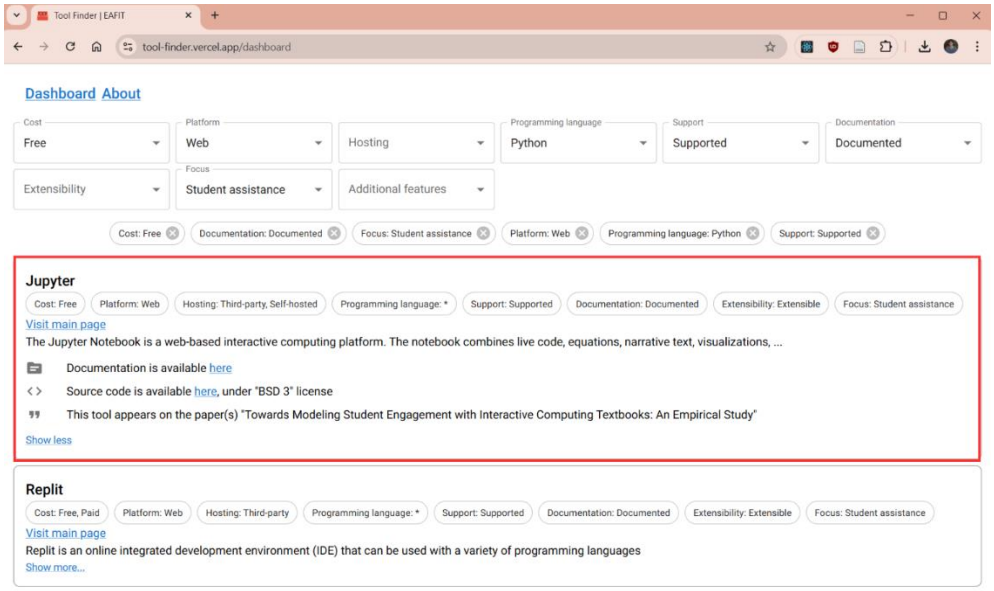

Figura 31. Selección de una herramienta para trabajo en clases

Por otra parte, para seleccionar la aplicación que nos ayudaría con la administración de los trabajos y prácticas externos a las clases identificamos que requeríamos lo siguiente: (i) una aplicación gratuita, ya que hay una amplia oferta de herramientas gratuitas y no consideramos necesario incurrir en gastos adicionales; (ii) una aplicación web, para que los estudiantes puedan usarla sin tener que lidiar con configuraciones complejas en sus equipos; (iii) soporte para Python, dado que es el lenguaje que será utilizado en el curso; (iv) soporte y documentación adecuados, para que el equipo docente pueda resolver los problemas que surjan de manera eficiente; y (v) un enfoque en la retroalimentación automatizada, que permite ofrecer a los estudiantes una evaluación inmediata de su código, ayudándoles a comprender errores y mejorar sus habilidades sin depender exclusivamente de la revisión manual del docente. En un curso con una matrícula tan amplia, la retroalimentación automatizada es clave para mantener una atención oportuna para cada estudiante y, al mismo tiempo, reducir significativamente la carga laboral del docente, permitiéndole enfocar su tiempo en áreas de apoyo más específicas. No utilizamos filtros relacionados con el tipo de *hosting*, ya que contamos con la capacidad de alojar la aplicación y, además, este enfoque facilita su integración con los sistemas Moodle de la universidad. Tampoco consideramos el filtro de características adicionales ni el de extensibilidad, ya que no se requerían funciones avanzadas para este contexto.

La Figura 32 muestra las herramientas recomendadas tras aplicar el filtro correspondiente. La aplicación seleccionó cuatro opciones que cumplieron con nuestros criterios: (i) CodeRunner, un plugin de evaluación automática para Moodle que permite crear preguntas tipo examen basadas en programación, con soporte para múltiples lenguajes; (ii) Jutge.org, una plataforma de juez en línea para la práctica y evaluación de código; (iii) Questournament, una plataforma para competiciones de programación que permite la evaluación automática; y VPL Plugin, un plugin de Moodle que permite configurar entornos virtuales de programación. Finalmente, se optó por VPL Plugin debido a que, además de cumplir con todos los requisitos, ofrece integración nativa con Moodle y dispone de un sistema básico de detección de plagio y comparación de similitud de código entre estudiantes, características que no están presentes en CodeRunner ni Questournament.

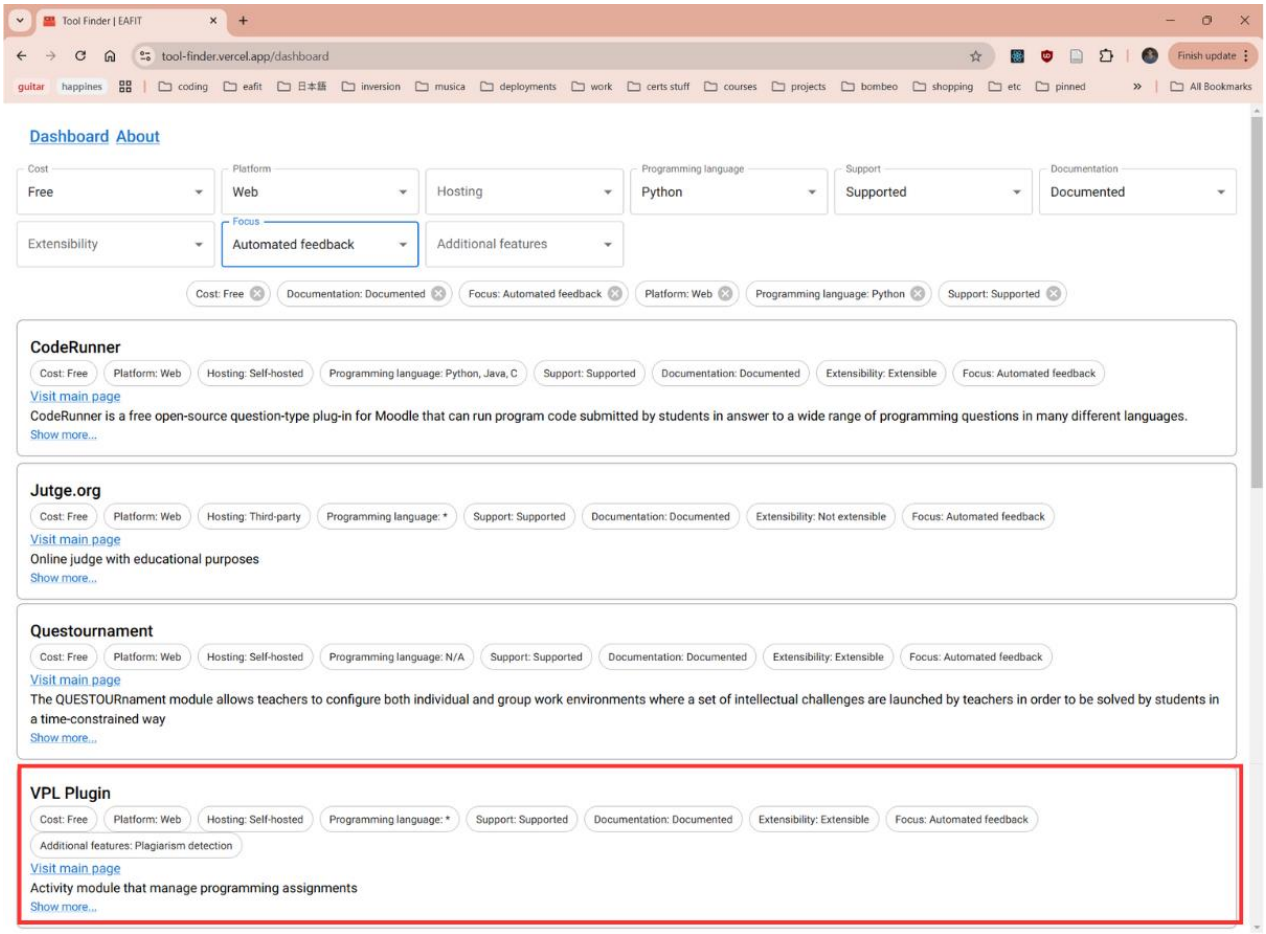

Figura 32. Selección de una herramienta para calificación automática de talleres

En el curso también se utilizó la aplicación PythonTutor como herramienta para la visualización de algoritmos en Python, aunque con menor énfasis que las herramientas VPL Plugin y Google Colab. Esto se debe a que PythonTutor no formó parte de un proceso específico del curso, sino que se empleó esporádicamente durante ciertas explicaciones. Además, se recomendó como herramienta complementaria para la depuración (debugging) y realización de pruebas. La Figura 33 ilustra las características de PythonTutor en Tool-Finder, destacando su naturaleza como una aplicación web gratuita diseñada específicamente para facilitar la comprensión visual de algoritmos y flujos de código.

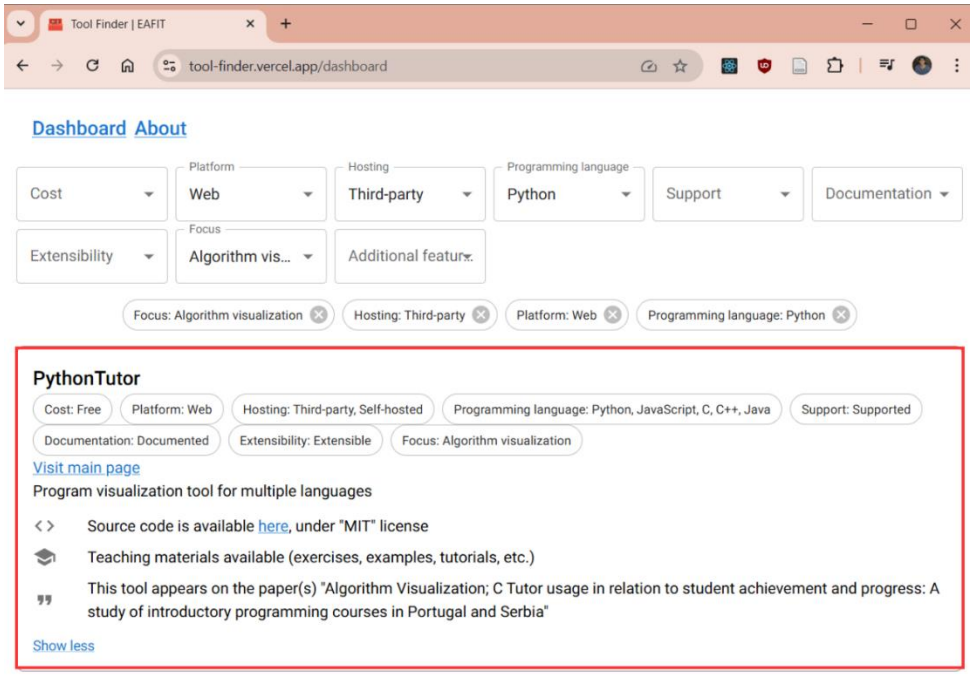

Figura 33. Características de PythonTutor en Tool-Finder

6.3 Metodología del curso

En esta sección se analizan los componentes metodológicos implementados en el curso Pensamiento Computacional, diseñados para optimizar el aprendizaje de los estudiantes, facilitar la enseñanza y gestionar eficazmente el curso. Estos componentes incluyen: (i) herramientas utilizadas, aplicaciones seleccionadas para apoyar el aprendizaje y la práctica de programación (subsección 6.3.1); (ii) clases presenciales, donde se implementaron estrategias en las sesiones en aula para garantizar un aprendizaje efectivo (subsección 6.3.2); (iii) monitorías, un apoyo proporcionado por monitores para reforzar conceptos y guiar a los estudiantes en sus actividades (subsección 6.3.3); y (iv) actividades evaluativas, que comprenden mecanismos de evaluación formativa y sumativa diseñados para medir el progreso y desempeño de los estudiantes a lo largo del curso (subsección 6.3.4).

A continuación, cada componente se describe en detalle, destacando su importancia y contribución al desarrollo del curso.

6.3.1 Herramientas utilizadas

Para la asignatura Pensamiento Computacional se emplearon dos plataformas principales: Google Colab y VPL Plugin, como se mencionó en la sección 6.2. Además, se utilizó de forma ocasional PythonTutor como herramienta de visualización de algoritmos, y también se recomendó a los estudiantes para para depuraran sus aplicaciones de manera interactiva.

A continuación, se describe cada aplicación y su uso específico dentro del curso, destacando su contribución al aprendizaje y a la práctica de programación.

Google Colab

Google Colab es es un servicio alojado de Jupyter Notebook que no requiere configuración y que ofrece acceso gratuito a recursos de computación. Es una herramienta en línea asociada a Google Drive, donde los estudiantes pueden programar desde el navegador, y los códigos desarrollados quedan almacenados en la cuenta de Google Drive. La Figura 34 muestra un ejemplo de código desarrollado en Google Colab.

Figura 34. Ejemplo de código desarrollado en Google Colab

Google Colab se utiliza en el curso como una herramienta integral para la realización de actividades prácticas en clase. Al ser el entorno que los estudiantes conocen y con el que se familiarizan desde el inicio, también se convierte en la plataforma predilecta para la presentación de evaluaciones importantes, como proyectos y exámenes.

VPL Plugin y Moodle

VPL (Virtual Programming Lab) es un plugin de Moodle que gestiona asignaciones de programación, destacándose por varias características importantes. Moodle, a su vez, es una plataforma de gestión del aprendizaje (LMS, por sus siglas en inglés) de código abierto que facilita la creación de entornos de aprendizaje en línea

VPL Plugin permite editar el código fuente de los programas directamente en el navegador y brinda a los estudiantes la posibilidad de ejecutar los programas de manera interactiva en la misma interfaz. Además, facilita la ejecución de pruebas para revisar el funcionamiento de los programas y ofrece una función de búsqueda de similitud entre archivos para detectar posibles plagios. Asimismo, permite configurar restricciones de edición, incluyendo la prohibición de pegar texto externo, con el fin de asegurar la integridad y originalidad del trabajo presentado.

El plugin VPL se integró en un sitio Moodle que estaba disponible en la universidad. Gracias a esta herramienta, los estudiantes tienen la posibilidad de programar directamente en el sitio, realizar ejercicios y recibir calificaciones automáticas. VPL facilita el proceso de aprendizaje al proporcionar un entorno interactivo y retroalimentación inmediata, lo cual fomenta la práctica constante de los conceptos de programación. En la Figura 35 se presenta un ejemplo de un código desarrollado utilizando VPL.

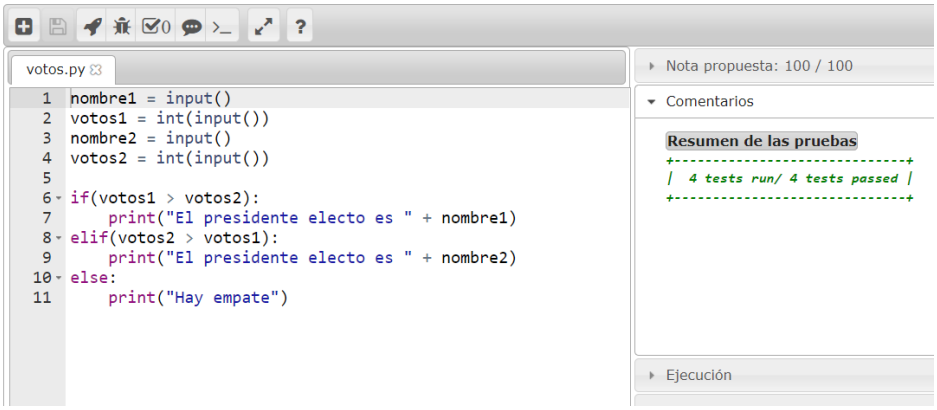

Figura 35. Ejemplo de código desarrollado en VPL Plugin

En el curso utilizamos Moodle para la gestión de calificaciones automatizadas mediante cuestionarios, así como para la integración del plugin VPL. El plugin VPL se utiliza principalmente para el desarrollo de talleres que son calificados automáticamente, lo que permite a los estudiantes practicar los contenidos semana a semana. Esta herramienta no solo facilita la evaluación continua y objetiva, sino que también fomenta el aprendizaje progresivo y autónomo de los estudiantes al ofrecer retroalimentación inmediata sobre sus ejercicios.

PythonTutor

PythonTutor es una plataforma educativa en línea que facilita el aprendizaje de la programación al permitir a los usuarios visualizar la ejecución de su código paso a paso. La Figura 36 muestra un ejemplo de visualización de ejecución de un programa en PythonTutor.

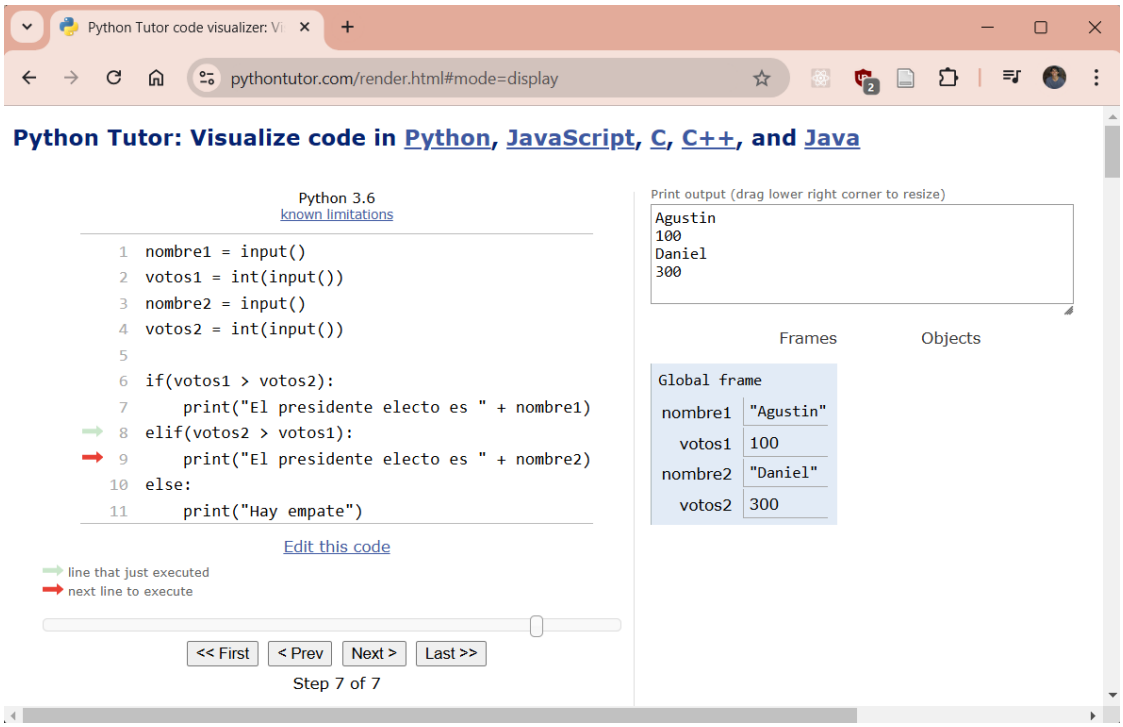

Figura 36. Ejemplo de visualización de un programa en PythonTutor

PythonTutor se utilizó en el curso de forma ocasional como una herramienta para la visualización de algoritmos, proporcionando a los estudiantes una representación gráfica del flujo de ejecución del código. Además, se recomendó su uso para que los estudiantes pudieran depurar sus aplicaciones de manera interactiva.

6.3.2 Clases presenciales

Las clases presenciales del curso de programación se abordaron haciendo uso de presentaciones en PowerPoint y realizando ejercicios prácticos utilizando Google Colab. De manera ocasional, también se empleó PythonTutor como herramienta de apoyo para la visualización de algoritmos; sin embargo, su uso no fue tan activo y no se realizó un seguimiento detallado de su implementación.

La Figura 37 presenta un ejemplo de una diapositiva explicativa tomada de las presentaciones utilizadas en la asignatura. Estas diapositivas servían para explicar conceptos y realizar ejemplos prácticos sobre su aplicación. Además, muchas diapositivas incluían el mensaje "codifíquelo", que incentivaba a los estudiantes a programar el código mostrado en pantalla, con el propósito de reforzar los conceptos y familiarizarse con la sintaxis del lenguaje de programación.

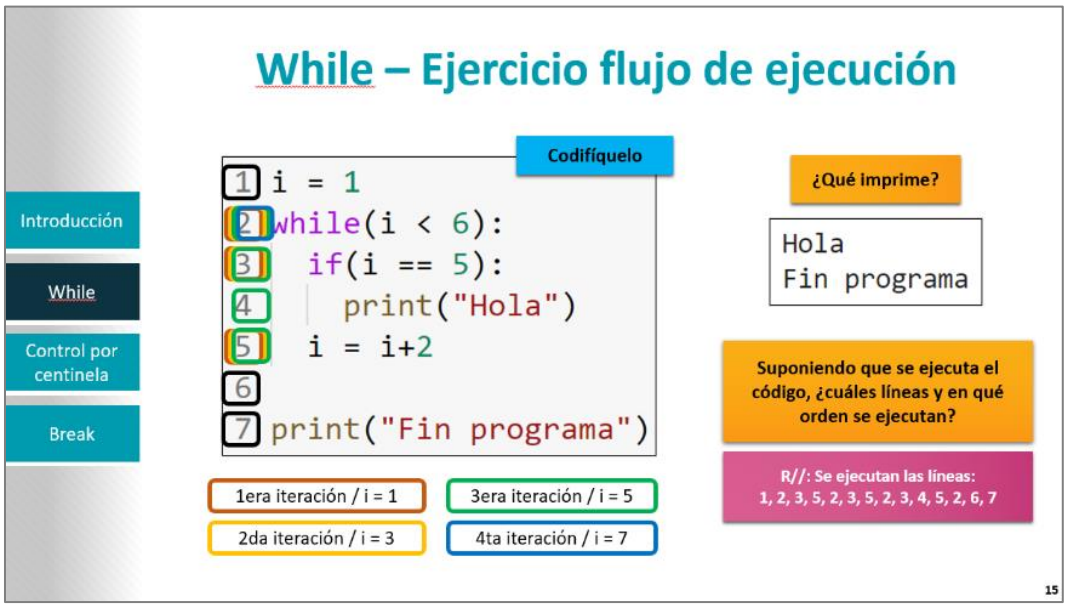

Figura 37. Ejemplo de una diapositiva explicativa de una de las presentaciones

Además de las diapositivas explicativas, las presentaciones incluían diapositivas enfocadas en promover el aprendizaje activo mediante ejercicios prácticos en clase. Estas diapositivas ofrecían instrucciones claras sobre la actividad, permitiendo a los estudiantes resolver los problemas planteados con el apoyo del docente y del monitor. La Figura 38 muestra un ejemplo de una diapositiva de ejercicio utilizada en clase.

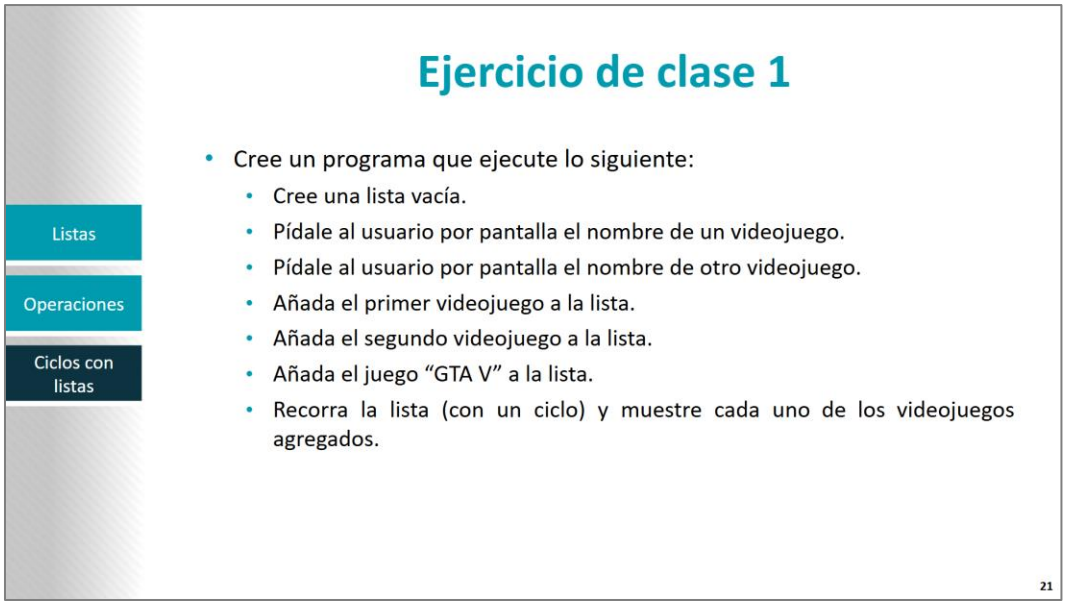

Figura 38. Ejemplo de una diapositiva de ejercicio de una de las presentaciones

6.3.3 Monitorías

El curso contó con un equipo de monitores, estudiantes de carreras relacionadas con la programación, que apoyaron al curso tanto dentro como fuera del aula. Durante las sesiones, asistieron a los docentes resolviendo dudas, mientras que fuera de clase ofrecieron horarios de atención para ayudar a los estudiantes con los talleres semanales. Este modelo de acompañamiento proporcionó soporte tanto dentro como fuera del aula, facilitando el aprendizaje para los estudiantes y reduciendo la carga laboral de los docentes.

6.3.4 Actividades evaluativas

En el curso se llevaron a cabo diversas actividades evaluativas que se pueden clasificar en dos tipos principales: actividades evaluativas formativas y actividades evaluativas sumativas. Las actividades formativas, como los quices y talleres, tenían como objetivo reforzar el aprendizaje mediante la práctica continua y la retroalimentación inmediata. Por otro lado, las actividades sumativas, que incluían proyectos y exámenes, estaban orientadas a evaluar el dominio global de los contenidos y las habilidades desarrolladas a lo largo del curso.

La Tabla 15 presenta la descripción de las actividades realizadas en el curso, junto con una breve descripción de sus objetivos, herramientas que las apoyaron y la metodología utilizada para su evaluación.

Actividad	**Objetivos**	**Herramientas de apoyo**	**Metodología**
Quices	Validar la comprensión de los contenidos de las clases e incentivar la asistencia a clases.	Moodle	Calificación automática con preguntas de tipo selección múltiple, emparejamiento de opciones y falso/verdadero.

Talleres	Motivar a los estudiantes a practicar activamente las temáticas vistas en clase.	Moodle VPL Plugin	Calificación automática de ejercicios de programación utilizando el plugin VPL.
Proyecto grupal	Aplicar los conocimientos adquiridos en el curso a retos más complejos y promover el trabajo en equipo.	Google Colab	Evaluación por el docente y retroalimentación formativa al código de los estudiantes.
Exámenes	Evaluar las habilidades de pensamiento computacional y programación adquiridas por los estudiantes.	Google Colab	Evaluación por el docente.

Tabla 15. Descripción de las actividades realizadas en el curso

Aunque el curso incluyó muchas actividades, la mayoría de ellas fueron automatizadas y varias se diseñaron con un nivel de dificultad bajo. A continuación, describimos a mayor profundidad cada una de las actividades evaluativas mencionadas anteriormente en el siguiente orden: (i) quices, (ii) talleres; (iii) proyecto grupal y (iv) exámenes.

I) Quices

Los quices se diseñaron con el objetivo de mantener el compromiso de los estudiantes, reforzar su aprendizaje y validar la comprensión de los contenidos de las clases. Los quices solo podían completarse en el aula, lo que contribuyó significativamente a la asistencia presencial, además de proporcionar retroalimentación constante tanto a estudiantes como a docentes, permitiendo identificar conceptos poco claros y tomar medidas correctivas como monitorías o repasos.

La implementación de los quices se llevó a cabo utilizando exclusivamente preguntas estándar de Moodle (selección múltiple, emparejamiento de opciones y falso/verdadero, por ejemplo). Para futuras iteraciones del curso, podría considerarse el uso de CodeRunner, lo que permitiría diversificar los tipos de preguntas al incluir ejercicios que involucren programación y ejecución de código. La Figura 39 muestra un ejemplo de tres preguntas utilizadas en los quices de la asignatura.

Figura 39. Ejemplo de preguntas de uno de los quices

II) Talleres

Los talleres del curso fueron diseñados para motivar a los estudiantes a practicar semanalmente las temáticas vistas en clase, considerando que cada tema se construye sobre los conocimientos adquiridos en semanas previas. Además de reforzar conceptos, los talleres demostraron la utilidad del pensamiento computacional y la programación aplicada a diversas áreas, presentando ejercicios con enunciados relacionados con música, literatura, derecho, ingeniería y más.

La Figura 40 muestra el enunciado de un ejercicio de uno de los talleres de la asignatura en conjunto con una posible solución para dicho enunciado. Estos ejercicios permitían intentos ilimitados para incentivar la práctica de los conceptos, siempre y cuando se respetara la fecha de entrega, que era estricta: los talleres debían completarse antes de la clase siguiente. Además, la plataforma verificaba automáticamente las respuestas enviadas mediante casos de prueba para evaluar si eran correctas, esto aseguraba que los estudiantes practicaran y consolidaran los temas antes de avanzar en el contenido del curso.

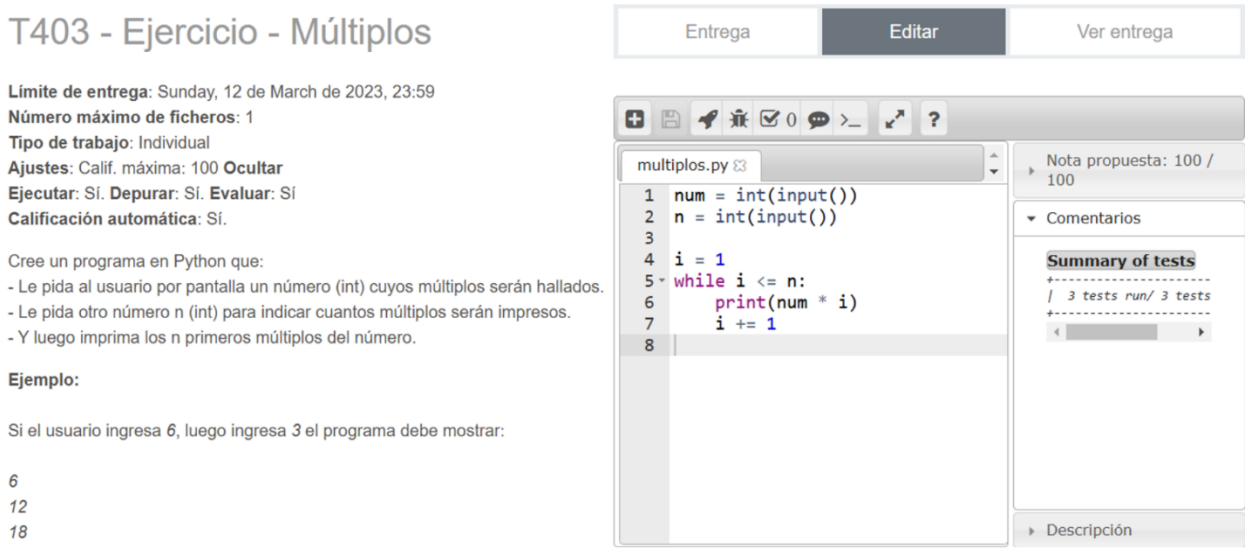

Figura 40. Ejemplo de un ejercicio de Taller junto con una posible solución

III) Proyecto grupal

El proyecto de la asignatura se diseñó con el objetivo de aplicar los conocimientos adquiridos en el curso a retos más complejos y promover el trabajo en equipo.

Para el desarrollo del proyecto, se proporcionó a los estudiantes un documento guía que incluía el contexto, un código base para iniciar el trabajo y una lista de actividades a realizar. El proyecto se llevó a cabo en Google Colab y los estudiantes tuvieron un espacio de sustentación presencial para presentar sus avances. La Figura 41 muestra el código base ofrecido en Google Colab, mientras que la Figura 42 presenta dos ejemplos de actividades a desarrollar a partir de dicho código.

```python
def mostrar(zombies):
  print("--Cantidad de zombies en Sabaneta--")
  print(zombies[0])
  print("--Cantidad de zombies en Envigado--")
  print(zombies[1])
  print("--Cantidad de zombies en Medellín--")
  print(zombies[2])
  print("--Cantidad de zombies en Itagüí--")
  print(zombies[3])

def disparar(zombies):
  posicionADisparar = int(input("Ingrese la posición de la ciudad a disparar (0, 1, 2 ó 3): "))
  zombies[posicionADisparar] = zombies[posicionADisparar]-10

print("--Creando los zombies en las ciudades--")
# zombies en Sabaneta, Envigado, Medellín, e Itagüí respectivamente
zombies = [40, 15, 23, 20]

while(True):
  mostrar(zombies)
  print("--Ingrese--")
  print("1) para disparar a zombies de una ciudad particular")
  accion = int(input())
  if(accion == 1):
    disparar(zombies)
```

Figura 41. Código base para el desarrollo del Proyecto

Actividad 1 (modificar la forma cómo se crean los zombies): La gobernadora quiere que la cantidad de zombies de cada ciudad, la ingrese el usuario por pantalla. Haga la modificación correspondiente para pedirle al usuario la cantidad de zombies de Sabaneta, Envigado, Medellín e Itagüí (y asigne esos valores a las posiciones 0, 1, 2 y 3 del arreglo de zombies).

Actividad 2 (modificar la forma cómo se ve la cantidad de zombies): a la gobernadora no le gusta la manera en la que se muestra la información de los zombies. En lugar de la forma actual, La gobernadora desea que la cantidad de zombies en cada ciudad se muestre en una cuadrícula como la siguiente (ustedes deberán modificar el código para hacerlo lucir de esa forma):

```
-----------------
 S: 40 | E: 15
-----------------
 M: 23 | I: 30
-----------------
```

El S: 40 representaría los zombies en Sabaneta, el E: 15 los zombies en Envigado, y así sucesivamente.

Figura 42. Ejemplo de dos actividades del Proyecto

IV) Exámenes

Los exámenes se diseñaron con el objetivo de contrastar las habilidades de pensamiento computacional y programación adquiridas por los estudiantes.

La Figura 43 muestra un ejemplo de pregunta práctica de exámen. Para estas preguntas se les solicitó a los estudiantes desarrollar la actividad utilizando Google Colab. Una vez completada,

debían copiar y pegar el código en la celda proporcionada debajo de la pregunta, lo que permitía su posterior revisión por parte del docente.

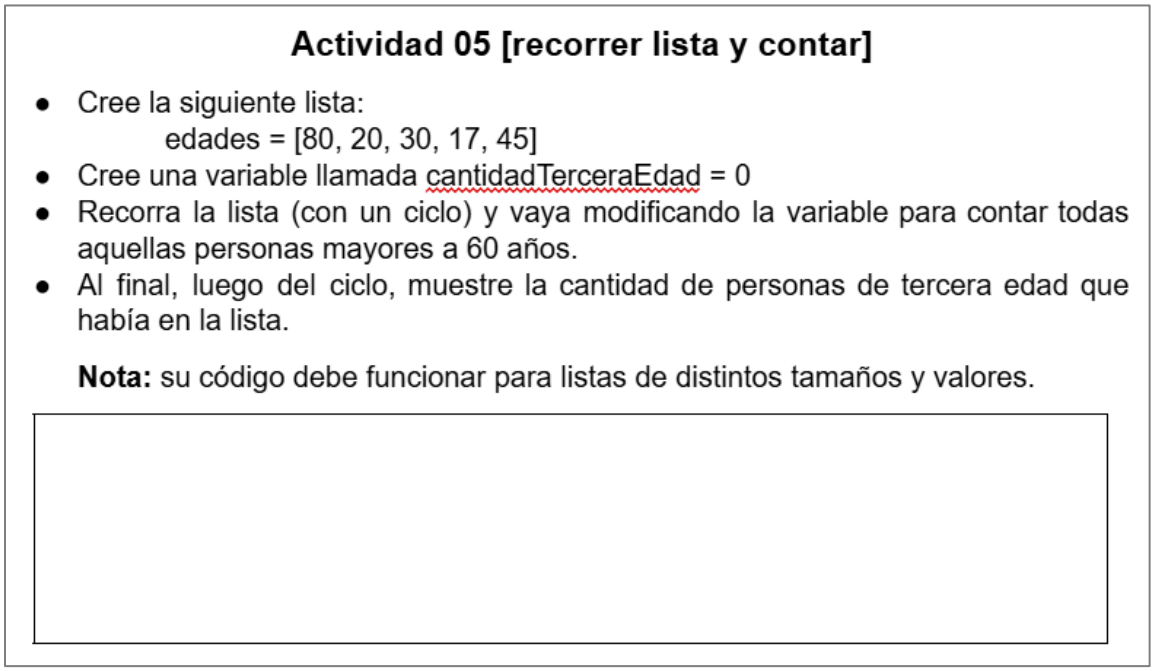

Figura 43. Ejemplo de pregunta práctica de exámen, generada siguiendo la guía

6.4 Discusión

El curso de Pensamiento Computacional dictado el semestre 2023-1 en la universidad EAFIT demuestra que la herramienta Tool-Finder puede ser utilizada para la selección de herramientas apropiadas para la enseñanza de un curso de programación introductoria. Además, valida que los elementos de contexto presentados en la sección 5.1 funcionan como criterios de selección de herramientas para cursos de programación en entornos de aplicación real.

A continuación, se analiza el impacto de las herramientas seleccionadas para el desarrollo del curso.

VPL Plugin y Moodle

La herramienta VPL plugin fue seleccionada, en parte, debido a la disponibilidad de la herramienta Moodle en la universidad. Ambas herramientas se utilizaron principalmente para proveer a los estudiantes con espacios para realizar práctica constante de los conocimientos adquiridos en clase, para mantenerlos enganchados al curso con talleres y para disminuir la carga laboral sobre el docente, dado que todas las actividades realizadas tanto en Moodle como VPL Plugin se calificaban de manera totalmente automática.

Moodle fue utilizado para la realización de los quices, donde se observó que los estudiantes no presentaron muchas dificultades en su uso, probablemente debido al diseño intuitivo de las preguntas y al uso de los formatos estándar de Moodle. Una limitación de la implementación de Moodle para los quices fue la falta de oportunidades para que los estudiantes evaluaran sus habilidades de programación directamente, lo cual podría haber enriquecido su experiencia de

aprendizaje. Se considera que los quices podrían mejorarse mediante el uso de herramientas como CodeRunner, especialmente para preguntas de tipo "completar código" u otros tipos de preguntas.

Google Colab

Google Colab se implementó ampliamente en el curso, utilizándose en las clases, el proyecto y los exámenes. Para garantizar su correcto uso, se dedicó una clase a explicar su configuración y funcionamiento y también se diseñó un tutorial al respecto, que se compartió a los estudiantes.

El impacto de la herramienta fue altamente positivo, ya que proporcionó a los estudiantes un entorno de programación en Python accesible, eliminando la necesidad de instalaciones complejas y permitiéndoles practicar desde cualquier computadora en cualquier momento. Una vez los estudiantes comprendían su funcionamiento, no presentaron ninguna dificultad para utilizar la plataforma. Para los docentes, Google Colab ofreció ventajas significativas al estandarizar el entorno de desarrollo y también al permitir compartir y solicitar código de manera sencilla a través de la plataforma.

PythonTutor

PythonTutor se utilizó en el curso principalmente como apoyo en explicaciones específicas durante las clases, donde su funcionalidad de depuración paso a paso resultó útil para los docentes y generó entusiasmo entre los estudiantes al visualizar ejemplos de manera detallada. Sin embargo, el uso directo de PythonTutor por parte de los estudiantes fue limitado. Una posible mejora en el curso sería integrar más esta herramienta, incluyendo enlaces que lleven a la visualización en PythonTutor de los ejemplos realizados en clase, lo que permitiría a los estudiantes interactuar directamente con la aplicación y adquirir experiencia práctica en la depuración de código.

Capítulo 7: Conclusiones y trabajo futuro

En esta sección se presentan las conclusiones de este estudio y las oportunidades de trabajo futuro que se identifican.

Conclusiones

Este trabajo se desarrolló siguiendo una estructura guiada por los objetivos de investigación definidos en el capítulo 3.

Para abordar el objetivo O1 identificamos las herramientas actuales utilizadas en la implementación de laboratorios de programación a través de un SMS. Como resumen encontramos que: (i) las herramientas más populares son Scratch, MOSS, iSnap, Alice, JUnit, PythonTutor, Lightbot y SnatchBot; (ii) los lenguajes más utilizados son Java, Python, C y C++; y (iii) existe un notable interés en el desarrollo de herramientas enfocadas en calificación automática, programación visual y detección de plagio. El proceso de identificación de las herramientas y la síntesis de su información se detalla en las subsecciones 4.1.4 y 4.1.5 de este documento.

Posteriormente, se analizaron las herramientas identificadas y se extrajeron datos sobre las motivaciones para su uso, lo que permitió identificar sus fortalezas y debilidades, cumpliendo así con el objetivo O2. Durante este proceso identificamos lo siguiente: (i) las herramientas de programación visual son atractivas para facilitar el aprendizaje de programación introductoria al reducir la carga cognitiva relacionada con la sintaxis; (ii) las herramientas de visualización de algoritmos son útiles para la comprensión de conceptos y la depuración de programas, aunque enfrentan retos técnicos que dificultan su mantenimiento a largo plazo; (iii) la literatura se enfoca principalmente en la detección de plagio, con pocas herramientas disponibles para su prevención; (iv) las herramientas de retroalimentación automática son ampliamente utilizadas para ofrecer retroalimentación rápida y precisa a los estudiantes, aunque presentan carencias en ciertas áreas críticas; y (v) las herramientas diseñadas para asistencia a estudiantes y la implicación suelen no estar disponibles al público o se encuentran abandonadas, lo que limita su utilidad. Este procedimiento se detalla en las secciones 4.2 y 4.3.

Una vez finalizado el mapeo de literatura se realizó un análisis de las herramientas y los cursos hallados en la literatura y se extrajo las características los elementos de contexto relevantes para la selección de herramientas. Algunos elementos de contexto extraídos fueron: (i) el costo de la herramienta, para distinguir entre aplicaciones gratuitas y de pago; (ii) la plataforma en que se ejecuta, diferenciando entre aplicaciones web y aquellas que requieren instalación; (iii) la necesidad de servidor o *hosting*, para identificar aplicaciones que deben ser hospedadas y mantenidas directamente; y (iv) los lenguajes de programación soportados, esenciales para seleccionar herramientas que se alineen con los objetivos del curso. Dichos elementos se describen a mayor profundidad en la sección 5.1, donde se cumple con el objetivo O3.

Para cumplir con el objetivo O4, se desarrolló una propuesta de implementación de laboratorios de programación basada en los contextos definidos previamente, con el fin de determinar las herramientas más adecuadas para un curso en particular. Esta propuesta se materializó en la aplicación Tool-Finder, la cual está diseñada para facilitar la búsqueda de las herramientas más adecuadas para cursos de programación haciendo uso de un sistema de filtros basado en elementos

de contexto relevantes, identificados a partir de la investigación realizada. El diseño, creación y demostración de funcionamiento de Tool-Finder se detallan en las secciones 5.2, 5.3 y 5.4.

Finalmente, para cumplir con el objetivo O5, se aplicó la propuesta a un curso universitario, evaluando su desempeño y resultados. En este proceso descubrimos que: (i) Tool-Finder es capaz de recomendar herramientas apropiadas para el caso de estudio propuesto, (ii) es posible establecer un enfoque sistémico para seleccionar la herramienta más adecuada para un curso de programación introductoria, basándose en elementos de contexto, y (iii) los elementos de contexto propuestos en esta investigación permiten seleccionar de manera asertiva herramientas de utilidad para dictar cursos de programación. Este análisis se detalla en el capítulo 6.

Trabajo futuro

Como trabajo futuro, se identifican varias áreas de investigación que se detallan a continuación:

- Explorar más elementos de contexto a partir de la literatura. Una identificación más granular de los elementos de contexto podría enriquecer el proceso de selección, ofreciendo criterios más precisos y mejorando la capacidad de ajustar las herramientas sugeridas a las necesidades específicas de los cursos.
- Complementar la identificación de herramientas con la inclusión de herramientas de IA que han tenido un auge en los últimos meses. Debido al momento en que se seleccionaron las publicaciones, la tecnología de la IA probablemente aún no estaba tan investigada, por lo cual no aparecen herramientas relacionadas en la investigación más allá de un par de *chatbots*.
- Ampliar el estudio de las herramientas no solo a herramientas enfocadas en cursos universitarios de primer semestre, sino a cursos posteriores, o analizar casos de uso en escuelas y secundaria. Investigar estas áreas podría aportar una visión más completa de las necesidades y adaptaciones necesarias en diferentes niveles de enseñanza de programación.
- Mejorar y ampliar las funcionalidades de la aplicación Tool-Finder para hacerla más robusta y facilitar su adopción en otras instituciones. Esto incluye realizar pruebas de usabilidad exhaustivas y analizar posibles funcionalidades que podrían optimizar su adaptabilidad a diferentes contextos educativos.
- Realizar pruebas comparativas con herramientas similares para evaluar su desempeño y efectividad en comparación con otras opciones disponibles. Esto incluye analizar la adopción de la herramienta por parte de grupos interesados y examinar cómo estas comparaciones pueden destacar sus fortalezas o indicar oportunidades de mejora.

Referencias

[1] M. S. Prokopyev, E. Z. Vlasova, T. V. Tretyakova, M. A. Sorochinsky, y R. A. Solovyeva, "Development of a Programming Course for Students of a Teacher Training Higher Education Institution Using the Programming Language Python", *Propósitos Represent.*, vol. 8, núm. 3, 2020, doi: 10.20511/pyr2020.v8n3.484.

[2] R. C. Ferguson, P. M. Leidig, y J. H. Reynolds, "Including a Programming Course in General Education: Are We Doing Enough?", *ISEDJ*, 2015.

[3] M. H. Baturay, "An Overview of the World of MOOCs", *Procedia - Soc. Behav. Sci.*, vol. 174, pp. 427–433, feb. 2015, doi: 10.1016/j.sbspro.2015.01.685.

[4] R.- ASALE y RAE, "Diccionario de la lengua española | Edición del Tricentenario", «Diccionario de la lengua española» - Edición del Tricentenario. Consultado: el 20 de noviembre de 2024. [En línea]. Disponible en: https://dle.rae.es/

[5] "Programación (Informática) - Qué es, usos, tipos y elementos", https://concepto.de/. Consultado: el 21 de noviembre de 2024. [En línea]. Disponible en: https://concepto.de/programacion/

[6] "Lenguaje de programación - Definicion.de", Definición.de. Consultado: el 21 de noviembre de 2024. [En línea]. Disponible en: https://definicion.de/lenguaje-de-programacion/

[7] Albérico Travassos Rosário y Joana Carmo Dias, "Learning Management Systems in Education: Research and Challenges", en *Advances in Educational Technologies and Instructional Design*, Nuno Geada y George Leal Jamil, Eds., IGI Global, 2022, pp. 47–77. doi: 10.4018/978-1-6684-4706-2.ch003.

[8] J. Waite y S. Sentance, "Teaching programming in schools: A review of approaches and strategies", 2021.

[9] J. H. H. Vasconcelos, "ESTUDIO TÉCNICO PARA LA IMPLEMENTACIÓN DEL SERVICIO DE INTERNET, EN EL LABORATORIO DE COMPUTACIÓN DE LA FACULTAD DE CIENCIAS ECONÓMICAS", UNIVERSIDAD DE SAN CARLOS DE GUATEMALA, Guatemala, 2004.

[10] M. Cardoso, R. Barroso, A. Vieira De Castro, y Á. Rocha, "VIRTUAL PROGRAMMING LABS IN THE COMPUTER PROGRAMMING LEARNING PROCESS, PREPARING A CASE STUDY", presentado en International Conference on Education and New Learning Technologies, Barcelona, Spain, mar. 2017, pp. 7146–7155. doi: 10.21125/edulearn.2017.2704.

[11] "VPL - Virtual Programming Lab - Home". Consultado: el 20 de noviembre de 2024. [En línea]. Disponible en: https://vpl.dis.ulpgc.es/

[12] K. Peffers, T. Tuunanen, M. A. Rothenberger, y S. Chatterjee, "A Design Science Research Methodology for Information Systems Research", *J. Manag. Inf. Syst.*, dic. 2007, doi: 10.2753/MIS0742-1222240302.

[13] K. Petersen, R. Feldt, S. Mujtaba, y M. Mattsson, "Systematic mapping studies in software engineering", en *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, en EASE'08. Swindon, GBR: BCS Learning & Development Ltd., jun. 2008, pp. 68–77.

[14] Adidah Lajis, Shahidatul Arfah Baharudin, Diyana Ab Kadir, Nadilah Mohd Ralim, Haidawati Mohamad Nasir, y Normaziah Abdul Aziz, "A review of techniques in automatic programming assessment for practical skill test", *J. Telecommun. Electron. Comput. Eng.*, 2018.

[15] Zahid Ullah, Adidah Lajis, Mona M. Jamjoom, Abdulrahman H. Altalhi, Ahmed A. Al-Ghamdi, y Farrukh Saleem, “The effect of automatic assessment on novice programming: Strengths and limitations of existing systems”, *Comput. Appl. Eng. Educ.*, 2018, doi: 10.1002/cae.21974.
[16] Hieke Keuning, Johan Jeuring, y Bastiaan Heeren, “A Systematic Literature Review of Automated Feedback Generation for Programming Exercises”, *ACM Trans. Comput. Educ.*, 2019, doi: 10.1145/3231711.
[17] Ibrahim Albluwi, “Plagiarism in Programming Assessments: A Systematic Review”, *ACM Trans. Comput. Educ.*, 2019, doi: 10.1145/3371156.
[18] Matija Novak, Mike Joy, y Dragutin Kermek, “Source-code Similarity Detection and Detection Tools Used in Academia: A Systematic Review”, *ACM Trans. Comput. Educ.*, 2019, doi: 10.1145/3313290.
[19] Andrew Luxton-Reilly *et al.*, “Introductory programming: a systematic literature review”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2018, doi: 10.1145/3293881.3295779.
[20] I. Eteng, S. Akpotuzor, S. O. Akinola, y I. Agbonlahor, “A review on effective approach to teaching computer programming to undergraduates in developing countries”, *Sci. Afr.*, vol. 16, p. e01240, jul. 2022, doi: 10.1016/j.sciaf.2022.e01240.
[21] A. L. S. O. De Araujo, W. L. Andrade, y D. D. Serey Guerrero, “A systematic mapping study on assessing computational thinking abilities”, en *2016 IEEE Frontiers in Education Conference (FIE)*, Erie, PA, USA: IEEE, oct. 2016, pp. 1–9. doi: 10.1109/FIE.2016.7757678.
[22] Juha Sorva, Ville Karavirta, y Lauri Malmi, “A Review of Generic Program Visualization Systems for Introductory Programming Education”, *ACM Trans. Comput. Educ.*, 2013, doi: 10.1145/2490822.
[23] A. Lukkarinen, L. Malmi, y L. Haaranen, “Event-driven Programming in Programming Education: A Mapping Review”, *ACM Trans. Comput. Educ.*, vol. 21, núm. 1, pp. 1–31, mar. 2021, doi: 10.1145/3423956.
[24] Andrew Luxton-Reilly, Theresia Devi Indriasari, y Paul Denny, “Improving Student Peer Code Review Using Gamification”, *IFAC Symp. Adv. Control Educ.*, 2021, doi: 10.1145/3441636.3442308.
[25] Oscar Karnalim y Simon, “Common Code Segment Selection: Semi-Automated Approach and Evaluation”, *Tech. Symp. Comput. Sci. Educ.*, 2021, doi: 10.1145/3408877.3432436.
[26] Rossevine Artha Nathasya, Oscar Karnalim, y Mewati Ayub, “Integrating program and algorithm visualisation for learning data structure implementation”, *Egypt. Inform. J.*, 2019, doi: 10.1016/j.eij.2019.05.001.
[27] Oscar Karnalim, None Simon, William J. Chivers, y Billy Susanto Panca, “Educating Students about Programming Plagiarism and Collusion via Formative Feedback”, *ACM Trans. Comput. Educ.*, 2022, doi: 10.1145/3506717.
[28] Oscar Karnalim y Mewati Ayub, “A Quasi-Experimental Design to Evaluate the Use of PythonTutor on Programming Laboratory Session”, *Int. J. Online Eng. Ijoe*, 2018, doi: 10.3991/ijoe.v14i02.8067.
[29] Paul Denny, Jacqueline Whalley, y Juho Leinonen, “Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback”, *IFAC Symp. Adv. Control Educ.*, 2021, doi: 10.1145/3441636.3442309.
[30] Samiha Marwan, Anay Dombe, y Thomas W. Price, “Unproductive Help-seeking in Programming: What it is and How to Address it”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2020, doi: 10.1145/3341525.3387394.

[31] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, y Thomas W. Price, “The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2019, doi: 10.1145/3304221.3319759.
[32] Samiha Marwan, Ge Gao, Susan R. Fisk, Thomas W. Price, y Tiffany Barnes, “Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science”, *Int. Comput. Educ. Res. Workshop*, 2020, doi: 10.1145/3372782.3406264.
[33] José Carlos Paiva, Ricardo Queirós, José Paulo Leal, Jakub Swacha, y Filip Miernik, “Managing Gamified Programming Courses with the FGPE Platform”, *Inf.*, 2022, doi: 10.3390/info13020045.
[34] Elena Verdú, Luisa M. Regueras, María Jesús Verdú, José Paulo Leal, Juan Pablo de Castro, y Ricardo Queirós, “A distributed system for learning programming on-line”, *Comput. Educ.*, 2012, doi: 10.1016/j.compedu.2011.08.015.
[35] Damla Topallı y Nergiz Ercil Cagiltay, “Improving programming skills in engineering education through problem-based game projects with Scratch”, *Comput. Educ.*, 2018, doi: 10.1016/j.compedu.2018.01.011.
[36] Boban Vesin, Mirjana Ivanović, Aleksandra Klašnja-Milićević, y Zoran Budimac, “Protus 2.0: Ontology-based semantic recommendation in programming tutoring system”, *Expert Syst. Appl.*, 2012, doi: 10.1016/j.eswa.2012.04.052.
[37] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, y Steve Cooper, “Mediated transfer: Alice 3 to Java”, *Tech. Symp. Comput. Sci. Educ.*, 2012, doi: 10.1145/2157136.2157180.
[38] Anuradha Mathrani, Shelly Christian, y Agate Ponder-Sutton, “PlayIt: Game Based Learning Approach for Teaching Programming Concepts.”, *Educ. Technol. Soc.*, 2016.
[39] Greg L. Nelson, Benjamin Xie, Amy J. Ko, y Andrew J. Ko, “Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1”, *Int. Comput. Educ. Res. Workshop*, 2017, doi: 10.1145/3105726.3106178.
[40] Richard J. Lobb y Jenny Harlow, “Coderunner: a tool for assessing computer programming skills”, *ACM Inroads*, 2016, doi: 10.1145/2810041.
[41] Sihan Li, Xusheng Xiao, Blake Bassett, Tao Xie, y Nikolai Tillmann, “Measuring code behavioral similarity for programming and software engineering education”, *2016 IEEEACM 38th Int. Conf. Softw. Eng. Companion ICSE-C*, 2016, doi: 10.1145/2889160.2889204.
[42] Ramón Zatarain Cabada, María Lucía Barrón-Estrada, Francisco González-Hernández, Raúl Oramas Bustillos, y Carlos A. Reyes-García, “An affective and Web 3.0-based learning environment for a programming language”, *Telemat. Inform.*, 2017, doi: 10.1016/j.tele.2017.03.005.
[43] Giuseppina Polito y Marco Temperini, “A gamified web based system for computer programming learning”, *Comput. Educ. Artif. Intell.*, 2021, doi: 10.1016/j.caeai.2021.100029.
[44] Abejide Ade-Ibijola, Sigrid Ewert, y Ian Sanders, “Abstracting and Narrating Novice Programs Using Regular Expressions”, *Res. Conf. South Afr. Inst. Comput. Sci. Inf. Technol.*, 2014, doi: 10.1145/2664591.2664601.
[45] Firas Layth Khaleel, Noraidah Sahari, y Tengku Siti Meriam Tengku Wook, “An Empirical Study On Gamification For Learning Programming Language Website”, *J. Teknol.*, 2019, doi: 10.11113/jt.v81.11133.
[46] Gloria Yi-Ming Kao y C.-A. Ruan, “Designing and evaluating a high interactive augmented reality system for programming learning”, *Comput. Hum. Behav.*, vol. 132, p. 107245, jul. 2022, doi: 10.1016/j.chb.2022.107245.

[47] Fang-Chuan Ou Yang, Hui-Min Lai, y Yen-Wen Wang, “Effect of augmented reality-based virtual educational robotics on programming students’ enjoyment of learning, computational thinking skills, and academic achievement”, *Comput Educ*, 2023, doi: 10.1016/j.compedu.2022.104721.
[48] Vu Nguyen, Hai Hoang Dang, Nguyen-Kha Do, y Dan-Thu Tran, “Enhancing team collaboration through integrating social interactions in a Web-based development environment”, *Comput. Appl. Eng. Educ.*, 2016, doi: 10.1002/cae.21729.
[49] Feng Zhang, Lulu Li, Cong Liu, y Qingtian Zeng, “Flow Chart Generation-Based Source Code Similarity Detection Using Process Mining”, *Sci. Program.*, 2020, doi: 10.1155/2020/8865413.
[50] Diana Pérez-Marín, Raquel Hijón-Neira, Angel Velazquez-Iturbide, Celeste Pizarro, y Luís Carriço, “Game programming for improving learning experience”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2014, doi: 10.1145/2591708.2591737.
[51] Chyanna Wee, Kian Meng Yap, y Woan Ning Lim, “IProgVR: Design of A Virtual Reality Environment to Improve Introductory Programming Learning”, *IEEE Access*, 2022, doi: 10.1109/access.2022.3204392.
[52] Soroush Ghorashi y Carlos Jensen, “Jimbo: a collaborative IDE with live preview”, *IEEEACM Int. Conf. Connect. Health Appl. Syst. Eng. Technol.*, 2016, doi: 10.1145/2897586.2897613.
[53] Sidra Iftikhar, M. Elena Rodríguez, y Enric Mor, “Practice Promotes Learning: Analyzing Students’ Acceptance of a Learning-by-Doing Online Programming Learning Tool”, *Appl. Sci.*, 2022, doi: 10.3390/app122412613.
[54] Atef Chorfi, Djalal Hedjazi, Sofiane Aouag, y Djalleleddine Boubiche, “Problem-based collaborative learning groupware to improve computer programming skills”, *Behav. Inf. Technol.*, 2020, doi: 10.1080/0144929x.2020.1795263.
[55] Q. Ding y S. Cao, “RECT: A Cloud-Based Learning Tool for Graduate Software Engineering Practice Courses With Remote Tutor Support”, *IEEE Access*, vol. 5, pp. 2262–2271, 2017, doi: 10.1109/ACCESS.2017.2664070.
[56] Timotej Lazar, Aleksander Sadikov, y Ivan Bratko, “Rewrite Rules for Debugging Student Programs in Programming Tutors”, *IEEE Trans. Learn. Technol.*, 2018, doi: 10.1109/tlt.2017.2743701.
[57] Molly Q. Feldman, Yiting Wang, William E. Byrd, François Guimbretière, y Erik Andersen, “Towards answering ‘Am I on the right track?’ automatically using program synthesis”, *SPLASH-E*, 2019, doi: 10.1145/3358711.3361626.
[58] Kannika Daungcharone, Patcharin Panjaburee, y Krittawaya Thongkoo, “Using Digital Game as Compiler to Motivate C Programming Language Learning in Higher Education”, *IIAI Int. Conf. Adv. Appl. Inform.*, 2017, doi: 10.1109/iiai-aai.2017.77.
[59] Mohammed Serrhini, Serrhini Mohammed, Abdelaziz Aït Moussa, y Ait Moussa Abdelazziz, “WIDE an interactive Web integrated development environment to practice C programming in distance education”, *2013 1st Int. Conf. Port. Soc. Eng. Educ. CISPEE*, 2013, doi: 10.1109/cispee.2013.6701964.
[60] Ryoya Yoshimura, Kazunori Sakamoto, Hironori Washizaki, y Yoshiaki Fukazawa, “WOJR: A Recommendation System for Providing Similar Problems to Programming Assignments”, *Appl. Syst. Innov.*, 2022, doi: 10.3390/asi5030053.
[61] R. Cedazo, Cecilia E. García Cena, y Basil Mohammed Al-Hadithi, “A friendly online C compiler to improve programming skills based on student self-assessment”, *Comput. Appl. Eng. Educ.*, 2015, doi: 10.1002/cae.21660.

[62] Jan Vanvinkenroye, Christoph Grüninger, Claus-Justus Heine, y Thomas Richter, “A Quantitative Analysis of a Virtual Programming Lab”, *IEEE Int. Symp. Multimed.*, 2013, doi: 10.1109/ism.2013.88.
[63] Ali Alammary, Angela Carbone, y Judy Sheard, “Implementation of a smart lab for teachers of novice programmers”, *IFAC Symp. Adv. Control Educ.*, 2012.
[64] David S. Janzen, John Clements, y Michael Hilton, “An evaluation of interactive test-driven labs with WebIDE in CS0”, *Int. Conf. Softw. Eng.*, 2013, doi: 10.1109/icse.2013.6606659.
[65] Marini Abu Bakar, Mohd Isrul Esa, Norleyza Jailani, Muriati Mukhtar, Rodziah Latih, y Abdullah Mohd Zin, “Auto-marking System: A Support Tool for Learning of Programming”, *Int. J. Adv. Sci. Eng. Inf. Technol.*, 2018, doi: 10.18517/ijaseit.8.4.6416.
[66] Jianxiong Gao, Bei Pang, y Steven S. Lumetta, “Automated Feedback Framework for Introductory Programming Courses”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2016, doi: 10.1145/2899415.2899440.
[67] Clara Benac Earle, Lars-Åke Fredlund, y John Hughes, “Automatic Grading of Programming Exercises using Property-Based Testing”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2016, doi: 10.1145/2899415.2899443.
[68] Sara Mernissi Arifi, Ismail Nait Abdellah, Azeddine Zahi, y Rachid Benabbou, “Automatic program assessment using static and dynamic analysis”, *World Conf. Complex Syst.*, 2015, doi: 10.1109/icocs.2015.7483289.
[69] Jesennia Cardenas-Cobo, Amilkar Puris, Pavel Novoa-Hernández, José A. Galindo, y David Benavides, “Recommender Systems and Scratch: An Integrated Approach for Enhancing Computer Programming Learning”, *IEEE Trans. Learn. Technol.*, 2019, doi: 10.1109/tlt.2019.2901457.
[70] Milena Vujošević-Janičić y Filip Marić, “Regression verification for automated evaluation of students programs”, *Comput. Sci. Inf. Syst.*, 2020, doi: 10.2298/csis181220019v.
[71] Iaeng Chinedu, “Revision-Bot: A Chatbot for Studying Past Questions in Introductory Programming”, 2022.
[72] Bruno Baruque y Álvaro Herrero, “Self-Assessment Web Tool for Java Programming”, *CISIS-ICEUTE*, 2015, doi: 10.1007/978-3-319-19713-5_51.
[73] Soundous Zougari, Mariam Tanana, y Abdelouahid Lyhyaoui, “Validity of a graph-based automatic assessment system for programming assignments: human versus automatic grading”, *Int. J. Power Electron. Drive Syst.*, 2022, doi: 10.11591/ijece.v12i3.pp2867-2875.
[74] Νικόλαος Πέλλας, Nikolaos Pellas, y Efstratios Peroutseas, “Gaming in Second Life via Scratch4SL: Engaging High School Students in Programming Courses.”, *J. Educ. Comput. Res.*, 2016, doi: 10.1177/0735633115612785.
[75] Iaeng Chinedu Wilfred Okonkwo Member y Iaeng Abejide Ade-Ibijola Member, “Python-Bot: A Chatbot for Teaching Python Programming”, 2020.
[76] Dirson Santos de Campos y Deller James Ferreira, “Plagiarism detection based on blinded logical test automation results and detection of textual similarity between source codes”, *Front. Educ. Conf.*, 2020, doi: 10.1109/fie44824.2020.9274098.
[77] Stella Biderman y Edward Raff, “Fooling MOSS Detection with Pretrained Language Models”, *Int. Conf. Inf. Knowl. Manag.*, 2022, doi: 10.1145/3511808.3557079.
[78] Arthur-Jozsef Molnar, Simona Motogna, y Cristina Vlad, “Using static analysis tools to assist student project evaluation”, *Proc. 2nd ACM SIGSOFT Int. Workshop Educ. Adv. Softw. Eng. Artif. Intell.*, 2020, doi: 10.1145/3412453.3423195.

[79] Mark Anderson y Collette Gavan, “Engaging undergraduate programming students: experiences using lego mindstorms NXT”, *Conf. Inf. Technol. Educ.*, 2012, doi: 10.1145/2380552.2380595.
[80] Chris Johnson, “SpecCheck: automated generation of tests for interface conformance”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2012, doi: 10.1145/2325296.2325343.
[81] Luís M. Alves *et al.*, “C Tutor usage in relation to student achievement and progress: A study of introductory programming courses in Portugal and Serbia”, *Comput. Appl. Eng. Educ.*, 2020, doi: 10.1002/cae.22278.
[82] Marco Aedo López, Elizabeth Vidal Duarte, Eveling Gloria Castro Gutierrez, y Alfredo Paz Valderrama, “Teaching Abstraction, Function and Reuse in the first class of CS1: A Lightbot Experience”, *Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, 2016, doi: 10.1145/2899415.2925505.
[83] João Alberto Fabro, Matheus Biscaya Gutierrez, y Fernando Henrique Ratusznei Caetano, “FluxProg 2.0 - Teaching Introductory Programming Using Flowcharts with Real and Simulated Robots for The Brazilian Robotics Olympiad (OBR)”, *Lat. Am. Robot. Symp.*, 2022, doi: 10.1109/lars/sbr/wre56824.2022.9995844.
[84] Teemu Sirkiä y Juha Sorva, “How Do Students Use Program Visualizations within an Interactive Ebook”, *Int. Comput. Educ. Res. Workshop*, 2015, doi: 10.1145/2787622.2787719.
[85] Nianfeng Shi, Zhiyu Min, y Ping Zhang, “Effects of visualizing roles of variables with animation and IDE in novice program construction”, *Telemat. Inform.*, 2017, doi: 10.1016/j.tele.2017.02.005.
[86] Oka Kurniawan, Norman Tiong Seng Lee, y Christopher M. Poskitt, “Securing Bring-Your-Own-Device (BYOD) Programming Exams.”, *Tech. Symp. Comput. Sci. Educ.*, 2020, doi: 10.1145/3328778.3366907.
[87] Srinivasan Lakshminarayanan y N. J. Rao, “Improving Integrity in CS1 Course Using Formative Assessment and Version Control Tools”, *High. Educ. Future*, 2021, doi: 10.1177/23476311211031876.
[88] Leonardo Mariani y Daniela Micucci, “AuDeNTES: Automatic Detection of teNtative plagiarism according to a rEference Solution”, *ACM Trans. Comput. Educ.*, 2012, doi: 10.1145/2133797.2133799.
[89] Andrés M. Bejarano, Lucy García, y Eduardo E. Zurek, “Detection of source code similitude in academic environments”, *Comput. Appl. Eng. Educ.*, 2015, doi: 10.1002/cae.21571.
[90] Liia Butler, Geoffrey Challen, y Tao Xie, “Data-Driven Investigation into Variants of Code Writing Questions”, *Conf. Softw. Eng. Educ. Train.*, 2020, doi: 10.1109/cseet49119.2020.9206195.
[91] José Bacelar Almeida, Alcino Cunha, Nuno Macedo, Hugo Pacheco, y José Proença, “Teaching how to program using automated assessment and functional glossy games (experience report)”, *Proc ACM Program Lang*, 2018, doi: 10.1145/3236777.
[92] Luciana Benotti, Federico Aloi, Franco Bulgarelli, y Marcos J. Gomez, “The Effect of a Web-based Coding Tool with Automatic Feedback on Students’ Performance and Perceptions”, *Tech. Symp. Comput. Sci. Educ.*, 2018, doi: 10.1145/3159450.3159579.
[93] Mona Nabil Demaidi, Manar Qamhieh, y Asmaa Afeefi, “Applying Blended Learning in Programming Courses”, *IEEE Access*, 2019, doi: 10.1109/access.2019.2949927.
[94] Ali H. Alharbi, “Re-imagining Computer Laboratories for Teaching Introductory Programming Concepts Using Web-based Integrated Development Environments:

Opportunities and Challenges", *World Symp. Softw. Eng.*, 2022, doi: 10.1145/3568364.3568375.
[95] J. A. Fredricks, P. C. Blumenfeld, y A. H. Paris, "School Engagement: Potential of the Concept, State of the Evidence", *Rev. Educ. Res.*, vol. 74, núm. 1, pp. 59–109, mar. 2004, doi: 10.3102/00346543074001059.
[96] Nasser Giacaman y Giuseppe De Ruvo, "Bridging Theory and Practice in Programming Lectures With Active Classroom Programmer", *IEEE Trans. Educ.*, 2018, doi: 10.1109/te.2018.2819969.
[97] School of Computing, National University of Singapore y S. Halim, "VisuAlgo – Visualising Data Structures and Algorithms Through Animation", *Olymp. Inform.*, vol. 9, pp. 243–245, jul. 2015, doi: 10.15388/ioi.2015.20.
[98] Jordi Petit, Omer Gimenez, y Salvador Roura, "Jutge.org: an educational programming judge", *Tech. Symp. Comput. Sci. Educ.*, 2012, doi: 10.1145/2157136.2157267.
[99] David Cowden, April O'Neill, Erik Opavsky, Duran Ustek, y Henry M. Walker, "A C-based introductory course using robots", *Tech. Symp. Comput. Sci. Educ.*, 2012, doi: 10.1145/2157136.2157150.
[100] "Google Colab". Consultado: el 28 de noviembre de 2024. [En línea]. Disponible en: https://research.google.com/colaboratory/faq.html#whats-colaboratory