Transición entre lenguajes de programación visual y lenguajes de programación textual.

Alumno: Nicolás Gamarra

Tutoría técnica: Ing. Ricardo Moran

Profesora de Trabajo Final: Dra. Marcela Samela

Facultad de Tecnología Informática, UAI

Trabajo Final de carrera presentado para obtener el título de Licenciatura en Gestión de Tecnología Informática

Agosto, 2022

Resumen:

En la actualidad la programación posee un rol fundamental en un mundo que presencia una profunda transformación impulsada por el desarrollo de la cultura digital. El desarrollo de software ha cobrado cada vez más importancia dentro de distintos ámbitos, tales como la industria, la salud, la educación y otros temas sociales de relevancia. Su cobertura es amplia, dado que incluye desde aplicaciones que brindan servicios básicos hasta desarrollos involucrados en el diseño y uso de robots. Particularmente la integración de la robótica en el ámbito educativo reviste un gran impacto, ya que acerca este campo de conocimiento a los estudiantes buscando promover habilidades que les permitan resolver diversas problemáticas sociales, crear nuevas oportunidades y prepararse para su integración al mundo laboral. La programación educativa -en conjunto con específicos entornos integrados de desarrollo - inicialmente emergió como un recurso para adquirir habilidades programáticas y, al mismo tiempo, aprender ciencias, incluyendo aspectos relacionados con la mecánica y la electrónica. De igual modo, la programación en los últimos años continuó ganando protagonismo y relevancia en ámbitos de desarrollo social y económico. Por esta razón, y observando la trascendencia que posee en la educación, se determinó que la transición entre lenguajes de programación visuales y textuales durante el aprendizaje deben ser un objeto estudio en sí mismo. En este caso, la investigación se realizó a través de una revisión sistemática de la literatura, con el propósito de estudiar los aspectos estrechamente vinculados a la programación y las herramientas que permiten llevarla a cabo por medio de ambientes integrados de desarrollo.

Palabras clave: educación, entornos integrados de desarrollo, lenguajes visuales
, programación, robótica

Abstract:

Currently, programming plays a fundamental role in a world witness to a huge transformation brought about by the development of digital culture. Software development has become increasingly important in different fields, such as industry, health, education and relevant social issues. Its coverage is extensive, including applications which provide core services for developments related to robot design and usage. Particularly, the integration of robotics in the educational field has a great impact, since it brings this field of knowledge to students who seek to promote skills that allow them to solve various social problems, create new opportunities and prepare for their integration into the world of work. Educational programming connectedly with specific integrated development environments originally emerged as a resource for the acquisition of programming skills and also for science learning, this includes mechanical and electronics aspects. Moreover, programs have become increasingly important and relevant to social and economic development over the past few years. Due to this and in relation to its importance in education, we investigated the transition between visual and textual programming languages during learning as an object of study in itself, in this case through a systematic review of the literature, in order to study the closely related aspects of programming and the tools that enable it to be carried out through integrated development environments.

Keywords: education, integrated development environment, programming, robotic, visual languages

Dedicatoria y agradecimiento

Dedico este trabajo a mi familia, quienes siempre me han brindado todo su apoyo en cada etapa de la vida, y a Madelaine, que me acompañó y alentó durante la escritura de este trabajo final. Agradezco también a mi tutor Ing. Ricardo Moran por haberme guiado, ofreciendo su conocimiento técnico y científico.

Por último, quiero agradecer a la Dra. Marcela Samela, quien me ha motivado, guiado y marcado el camino para la finalización del presente trabajo.

Índice General

Tabla de contenido

Resumen: 2

Abstract: 3

Índice General 5

Índice de figuras 8

Índice de tablas 9

Capítulo 1 - Introducción 10

1.1 Introducción 10

1.2 Pregunta de investigación 11

1.3 Hipótesis. 11

1.4 Objetivo General 11

1.5 Objetivos Particulares 12

1.6 Estructura general del trabajo final 12

1.7 Metodología de Investigación 13

Capítulo 2 – Marco teórico 14

2.1 Lenguajes de programación. 14

2.1.1 Lenguajes de programación textuales. 17

2.1.1.1 Lenguajes de bajo nivel. 18

2.1.1.2 Lenguajes de alto nivel. 20

2.1.2 Lenguajes de programación visuales. 23

2.1.2.1 Clasificación de lenguajes visuales 23

2.2 Entornos integrados de desarrollo: definiciones, componentes y características. 25

2.2.1 Características principales. 27

2.2.2 Entornos integrados de desarrollo para la educación. 28

Capítulo 3 – Transición entre lenguajes de programación visuales y textuales. 33

2.1 La importancia de la transición entre lenguajes de programación 33

2.2 Razones por las que se producen discordancias entre lenguajes de programación. 35

2.2.1 Problemas en la enseñanza del pensamiento computacional. 35

2.2.2 Problemas de sintaxis ……37
2.3 Estrategias y herramientas para abordar la transición entre lenguajes de programación. ……38
2.3.1 Herramientas para abordar la transición ……38
2.3.2 Estrategias para abordar la transición entre lenguajes visuales y textuales. ……41
2.3.2.1 Estrategia basada en lenguajes de desarrollo híbridos ……41
2.3.2.2 Estrategia basada en bloques ……42
2.3.2.3 Estrategia basada en juegos …… 43
Capítulo 4 – Planificación. ……..43
4.1 Identificar la necesidad de revisión. ……44
4.2 Definir las preguntas de investigación que conducirán la búsqueda …… 44
4.3 Desarrollo protocolo de revisión ……45
4.3.1 Los términos de búsqueda ……46
4.3.2 Estrategia de búsqueda ……48
4.3.3 Criterios de inclusión y exclusión ……48
4.3.4 Procedimiento de selección de estudios ……50
4.3.5 Evaluación de calidad ……51
4.3.6 Estrategia de extracción de datos ……52
4.3.7 Obtención de datos ……53
4.4 Resultados obtenidos ……55
4.4.1 Análisis cuantitativo ……55
4.4.2 Análisis cualitativo ……55
Capítulo 5 – Ejecución ……..56
5.1 Ejecución de la estrategia de búsqueda. ……56
5.2 Proceso de selección de trabajos. ……57
5.3 Procesamiento y extracción de los datos ……58
5.3.1 Primera etapa …… 58
5.3.2 Segunda Etapa ……59
5.3.3 Tercera etapa …… 59
Capítulo 6 – Resultados y análisis de los datos …….. 65
5.1 Análisis de los resultados ……65
5.2 Análisis cuantitativo ……65
5.2.1 Análisis de los trabajos que presentaron herramientas de software. …… 67

5.2.2 Análisis de los trabajos que no presentaron herramientas de software 70
5.3 Análisis cualitativo 74
Capítulo 7 - Conclusión y trabajos futuros. 78
7.1 Conclusión 78
7.2 Líneas futuras de investigación 80
7.3Anexos.....81
Acrónimos 81
Referencias 82

Índice de figuras

FIGURA 1 61

FIGURA 2 68

FIGURA 3 69

FIGURA 4 70

FIGURA 5 71

FIGURA 6 72

FIGURA 7 73

FIGURA 8 73

FIGURA 9 74

FIGURA 10 75

FIGURA 11 76

FIGURA 12 77

FIGURA 13 78

FIGURA 14 79

Índice de tablas

TABLA 1 22

TABLA 2. 30

TABLA 3 40

TABLA 4 45

TABLA 5 47

TABLA 6 47

TABLA 7 48

TABLA 8 48

TABLA 9 49

TABLA 10 53

TABLA 11 62

TABLA 12 65

Capítulo 1 - Introducción

1.1 Introducción

Durante las últimas décadas los entornos de programación educativa comenzaron a estar en auge, transformándose en una herramienta útil para la enseñanza y el aprendizaje de la programación. Por dicho motivo numerosas instituciones educativas la incluyen la disciplina como una asignatura dentro de su plan de estudios, impulsando que desde más jóvenes los estudiantes comiencen a programar.

En este contexto los entornos integrados de desarrollo adquieren un rol fundamental, proporcionando facilidades para la programación de software y permitiendo maximizar la productividad de quienes ya los utilizan.

En la actualidad, para programar profesionalmente es necesario conocer lenguajes de programación textuales, pero para utilizarlos se necesitan también ciertos conocimientos matemáticos, capacidad de abstracción y buena concentración. El aprendizaje de un lenguaje de programación es complejo, por lo cual, para reducir dicha complejidad en las últimas décadas a surgieron los Entornos Visuales de Programación (EVP).

En los EVP, los elementos del lenguaje de programación se disponen de manera gráfica, (con menores niveles de complejidad) buscando mediante la inclusión de distintas propuestas mejorar la comprensión de los conceptos principales, al mismo tiempo que se facilita la enseñanza de habilidades programáticas y se fomenta la motivación de los estudiantes dentro de los distintos niveles del ciclo educativo.

Así surge entonces el interés en conocer herramientas y estrategias que permitan abordar la transición entre lenguajes de programación visuales y lenguajes de programación textuales para cubrir esta necesidad educativa.

1.2 Pregunta de investigación

¿Cuál es la estrategia educativa más utilizada para realizar la transición entre lenguajes de programación visuales y textuales surgida en los últimos 30 años?

1.3 Hipótesis.

En esta investigación planteamos la siguiente hipótesis:

Dentro del contexto educativo, el desarrollo de estrategias que implementan un modelo híbrido de desarrollo, basado en componentes visuales y textuales, ha sido la estrategia más utilizada desde el año 1990 a la actualidad.

La hipótesis planteada permite ser corroborada en función a la cantidad de estudios académicos relevantes sobre herramientas que posibilitan la transición entre lenguajes textuales y lenguajes visuales, que emergieron durante las últimas décadas.

1.4 Objetivo General

En este trabajo se realiza una revisión sistemática de la literatura para identificar, evaluar e interpretar toda la información disponible y concerniente a la siguiente pregunta de investigación: "¿Cuál es la estrategia educativa más utilizada para realizar la transición entre lenguajes de programación visuales y textuales"?

El abordaje de esta revisión sistemática nos permitirá establecer de forma rigurosa la situación actual sobre esta temática y posibilitará tanto resumir la evidencia existente vinculada a los entornos integrados de desarrollo como indagar en las estrategias para llevar a cabo la transición entre los paradigmas de programación visuales y textuales.

1.5 Objetivos Particulares

El objetivo particular de este trabajo es, verificar la literatura disponible en referencia a los entornos integrados de desarrollo; verificar las tecnologías que emplean; identificar si los mismos son visuales o textuales, o bien procedurales, funcionales u orientados a objetos; si se trata de hardware abierto o propietario y bajo dichas premisas cuáles de estos se utilizan para el desarrollo de robótica educativa.

1.6 Estructura general del trabajo final

- Capítulo 1: Introducción
- Capítulo 2: Marco Teórico.
- Capítulo 3: Transición entre lenguajes de programación textuales y visuales.
- Capítulo 4: Planificación.
- Capítulo 5: Ejecución.
- Capítulo 6: Resultados y análisis de los datos.
- Capítulo 7: Conclusión y trabajos futuros.

1.7 Metodología de Investigación

La metodología de investigación elegida para el presente trabajo es la revisión sistemática de la literatura propuesta por Kitchenham y Charters (2009).

Se identificó la necesidad para llevar a cabo la RSL y se formularon preguntas de investigación para sustentar la hipótesis del presente trabajo.

El objetivo de la metodología utilizada es analizar trabajos académicos relacionados con el tema *transición de lenguajes visuales a lenguajes textuales* haciendo énfasis en las herramientas, métodos y técnicas propuestas para llevarla a cabo. Las preguntas de investigación serán la guía, mientras que el análisis de la información encontrada buscará satisfacer los interrogantes propuestos.

Según Kitchenham y Charters (2009), una revisión sistemática de la literatura, a menudo referida como revisión sistemática, es un medio para identificar, evaluar e interpretar toda la información pertinente y disponible a una pregunta de investigación en particular, área, temática o fenómeno de interés. Los estudios individuales que contribuyen a una revisión sistemática se denominan estudios primarios mientras que una revisión sistemática es una forma de estudio secundario.

La búsqueda y recuperación de información en repositorios se trabajará de forma iterativa, para lo cual se determinarán términos de búsqueda relacionados con las preguntas de investigación y se emplearán a su vez sinónimos que contribuyan a obtener la cadena de búsqueda más concisa posible.

Seleccionadas las fuentes y tras el resultado surgido por la ejecución de la cadena de búsqueda con los términos definidos, se escogerán los estudios primarios mediante la aplicación de criterios de exclusión e inclusión. Obtenidos los estudios relevantes se continuará con la

extracción y análisis de los datos para dar respuesta a las preguntas de investigación formuladas. Finalmente se elaborarán las conclusiones de la revisión sistemática.

Los estudios irrelevantes serán descartados mediante criterios de inclusión y exclusión, examinando cada estudio para decidir su pertinencia. En una primera instancia se tomarán el título, el resumen y las palabras clave como criterios iniciales, mientras que en una segunda instancia se revisará el contenido completo de cada potencial estudio.

Capítulo 2 – Marco teórico.

En el segundo capítulo se comienza conceptualizando de manera general lenguajes de programación y entornos integrados de desarrollo desde el enfoque de varios autores. Se definen, a su vez, los conceptos más utilizados y sus principales características para luego presentar distintas clasificaciones de estos. El foco principal se encuentra en la variedad de entornos de desarrollos que existen y en los elementos principales que se comparten y/o colaboran.

2.1 Lenguajes de programación.

Los lenguajes para la programación permiten a los desarrolladores indicarle de manera precisa a una computadora cómo debe trabajar u operar con los datos que se le provean, posibilitando el almacenamiento y la transmisión de ellos. También permiten indicar que acción se deberá realizar según la circunstancia que se presente o tarea que se deba resolver.

Según Joyanes (2008) un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se denominan

programación. Entonces, los lenguajes utilizados para escribir programas de computadoras son los lenguajes de programación y los programadores son los escritores y diseñadores de programas. (p.35)

Con motivo de facilitar la tarea de los programadores, los lenguajes de programación buscan ser semejantes a un lenguaje humano o natural, ya que posibilitan la comunicación entre el programador y la computadora, partiendo de un entendimiento común del lenguaje y utilizando conjuntos de símbolos, palabras claves, reglas semánticas y sintácticas que posibiliten la creación de programas.

Gabbrielli y Martini (2010) señalan que los algoritmos que se desean ejecutar deben representarse utilizando las instrucciones de un lenguaje de programación. Este lenguaje se definirá formalmente en términos de una sintaxis específica y una semántica precisa que, sin importar su naturaleza, permitirá utilizar un conjunto finito de constructos llamados *instrucciones* para construir programas.

Cabe considerar, por otra parte, que para que los programadores puedan interactuar con la computadora y lleven a cabo la compleja tarea de programar, se debe mencionar al verdadero responsable de estas acciones: el software del sistema.

El software del sistema es el conjunto de programas indispensables para que la computadora funcione, denominado también *software de aplicación*. Estos programas son el sistema operativo, los editores de texto, los compiladores/intérpretes (lenguajes de programación) y los programas de utilidad (Joyanes, 2008).

El software de aplicación tiene como función principal asistir y ayudar a un usuario de un computador para ejecutar tareas específicas. Los programas de aplicación se pueden desarrollar con diferentes lenguajes y herramientas de software (Joyanes, 2008).

Dentro de este marco es preciso comentar, también, cuándo se originaron los lenguajes de programación. Hacia la década de 1950 surgieron lenguajes que se conocen como Lenguajes Procedimentales (por procedimientos) Un programa en un lenguaje procedimental es un conjunto de instrucciones o sentencias (Joyanes, 2008).

Con el pasar de los años, los lenguajes de programación fueron evolucionando. Hacia la década de 1970, desde el punto de vista del software, el procesamiento por lotes de instrucciones dio paso a un enfoque más interactivo en el que el usuario, mediante una terminal, podía interactuar directamente con la ejecución, algo que resultaba imposible con lenguajes de la década anterior. Por consiguiente, esta interacción más directa con la computadora dio lugar al nacimiento de nuevos paradigmas de programación: la *programación orientada a objetos* y la *programación declarativa* (Gabbrielli y Martini, 2010).

Durante la década de 1980, el desarrollo de las primeras computadoras personales por parte de grandes compañías como Apple, IBM o Microsoft ocasionó un cambio completo en el papel de los lenguajes de programación. Sucede, pues, que el desarrollo de sistemas para uso personal que proporcionan interfaces gráficas fáciles de utilizar trajo consigo la necesidad de desarrollar sistemas gráficos fácilmente interactivos que pudieran administrar interfaces de ventana. Estos sistemas han proporcionado, por lo tanto, un área de aplicación ideal para lenguajes orientados a objetos. De hecho, los lenguajes de la década de 1980 se concibieron como orientados a objetos, experimentando un desarrollo significativo durante este periodo (Gabbrielli y Martini, 2010).

Desde la década de 1990 y hasta la actualidad, atestiguamos el surgimiento de Internet y el World Wide Web, dos herramientas que han cambiado profundamente muchos aspectos de la computación y, por tanto, también los lenguajes de programación. La evolución de dichos lenguajes está presente en todos los espectros de un sistema informático, tanto desde el protocolo

de comunicación, que posibilita la interacción entre computadoras separadas a miles de kilómetros de distancia, como también lenguajes para la presentación de páginas web y tratamiento de datos (Gabbrielli y Martini, 2010).

En relación con lo mencionado anteriormente, y en términos de la profunda evolución de la tecnología -tal como expresa Babkin (2012) con el crecimiento del hardware de gráficos para las computadoras y los avances del software para renderizarlos- se hizo posible representar los datos en forma bidimensional o tridimensional, contribuyendo también al avance de la programación visual.

De esta manera, gracias a los avances de la tecnología, en la actualidad encontramos distintos tipos de lenguajes de programación, destacando entre sus principales exponentes los lenguajes textuales y los lenguajes visuales.

2.1.1 Lenguajes de programación textuales.

La programación mediante lenguajes textuales o basada en texto se refiere al conjunto de símbolos, caracteres e instrucciones que les permiten a los programadores realizar diversas operaciones basándose en reglas ya establecidas del propio lenguaje. A su vez, este conjunto de sentencias es interpretado y traducido por los procesadores a lenguaje máquina.

Según Joyanes (2008) la información en formato de texto se representa mediante un código en el que cada uno de los distintos símbolos del texto, tales como letras del alfabeto o signos de puntuación, se asignan a un único patrón de bits. El texto se representa como una cadena larga de bits en la cual los sucesivos patrones representan los sucesivos símbolos del texto original.

Esta acción es posible, tal como se explicó anteriormente, gracias a programas intérpretes y compiladores. Los primeros adaptan las instrucciones mientras se van detectando y

los segundos traducen los programas escritos en un lenguaje de programación a lenguaje máquina.

Debe señalarse que la programación textual requiere que el desarrollador conozca de la sintaxis, semántica y gramática específica del lenguaje que se quiera utilizar, por lo que previamente se requieren tanto conocimientos de lectura y escritura como también poseer un cierto grado de abstracción. Estos elementos resultan de vital relevancia ya que de acuerdo a las edades de los estudiantes será posible o no lograr el aprendizaje y la enseñanza de un lenguaje de programación textual.

En relación con lo mencionado en el párrafo anterior, debe señalarse que los programas que implementan aplicaciones y sistemas de software constan de miles, cientos de miles o incluso millones de líneas de código. Independientemente del lenguaje utilizado, se especificará un conjunto de reglas para formar programas válidos en ese lenguaje (Ghezzi, 1996).

Dentro de este orden ideas, es necesario tener claro que los lenguajes de programación poseen dos componentes principales: la sintaxis y la semántica. La sintaxis es el conjunto de reglas que definen cómo se escribe e interpreta un programa (Parveen y Fatima, 2016), mientras que la semántica específica “el significado” de cualquier programa sintácticamente válido y escrito en un lenguaje, definiendo el sentido o coherencia de los programas en los cuales su sintaxis sea correcta. (Ghezzi, 1996).

2.1.1.1 Lenguajes de bajo nivel.

Se denominan lenguajes de bajo nivel a aquellos lenguajes cuyas máquinas abstractas están muy cerca de coincidir con la máquina física. Estos nacieron a fines del año 1940, y se utilizaron para programar las primeras computadoras pero eran extremadamente difíciles de usar. Para

indicar instrucciones en estos lenguajes era necesario tener en cuenta las características físicas de la máquina, nociones completamente irrelevantes para el algoritmo. Estas debían tenerse en cuenta al escribir programas o al codificar algoritmos y, por lo tanto, cuando se habla genéricamente de "Lenguaje maquina", se está refiriendo al lenguaje (de bajo nivel) de una máquina física (Gabbrielli y Martini, 2010, p. 5).

Como señalan Gonzalez et al. (2012) se trata del lenguaje de más bajo nivel, que a través de un sistema binario basado en ceros y unos, muestra datos e instrucciones, y el elemento ejecutor es la circuitería misma de la computadora.

En esta última definición, la noción de un lenguaje de bajo nivel se aborda como un lenguaje primitivo que está formado por una combinación de dígitos binarios comprendidos entre los valores 0 y 1 (código binario) y que al combinarse se transforman en una secuencia de bits que especifican una determinada instrucción interpretada, procesada y ejecutada por el hardware.

Desde el punto de vista de Gabbrielli y Martini (2010), con el tiempo se comprendió que para aprovechar al máximo el poder de la computadora era necesario desarrollar formalismos adecuados que estuvieran lejos de ser un lenguaje máquina y más cerca del lenguaje natural del usuario. Un primer paso en esta dirección fue la introducción de lenguajes ensambladores. Estos lenguajes pueden ser vistos como representaciones simbólicas del lenguaje máquina y, de hecho, existe una correspondencia uno a uno entre un gran número de instrucciones en lenguaje máquina y códigos en lenguaje ensamblador. Los programas escritos en lenguaje ensamblador son traducidos a programas escritos en lenguaje máquina por un programa llamado *ensamblador* (p. 415).

El lenguaje ensamblador es un lenguaje más cercano al hombre, donde cada instrucción se corresponde con una instrucción en el lenguaje máquina. El programa ensamblador es el

encargado de traducir un programa de usuario -o programa "fuente" escrito con mnemónicos- a un programa de lenguaje de máquina o programa "objeto" que la microcomputadora puede ejecutar. La entrada del ensamblador es un programa fuente y su salida es un programa objeto (Knaggs y Welsh, 2019, p. 4).

Los lenguajes de bajo nivel se encuentran estrechamente relacionados a las instrucciones del hardware, facilitando el control de estos y estando limitados por los componentes físicos de las computadoras que lo soportan. Son utilizados generalmente para desarrollar tareas y funciones de los sistemas operativos, controladores o aplicaciones específicas ligadas al hardware y el manejo de estos lenguajes requiere un cierto grado de especialización de los programadores.

2.1.1.2 Lenguajes de alto nivel.

Los lenguajes de programación de alto nivel son aquellos que apoyan el uso de construcciones que utilizan mecanismos de abstracción apropiados para asegurar que sean independientes de las características físicas del ordenador. Los lenguajes de alto nivel son adecuados para expresar algoritmos en formas que resultan relativamente fáciles de entender para el usuario humano (Gabbrielli y Martini, 2010, p. 5).

Un lenguaje de programación que se encuentra muy cerca del lenguaje natural y es una abstracción del hardware y el firmware adjunto de la máquina se conoce como *lenguaje de programación de alto nivel* (Parveen y Nazish, 2016).

Según Gabbrielli y Martini (2010) el verdadero salto en calidad se logró en la década de 1950 con la introducción de lenguajes de alto nivel, también llamados *lenguajes de tercera generación*. Estos eran diseñados como lenguajes abstractos que ignorarían las características físicas de la computadora y, en cambio, estaban preparados para expresar algoritmos de una

manera que resultaba relativamente fácil para el usuario humano. Entre los primeros intentos se encuentran algunos formalismos que permitieron el uso de la notación simbólica para indicar expresiones aritméticas. Las expresiones así codificadas podrían luego traducirse automáticamente a instrucciones ejecutables por la máquina (p. 415).

Dichos lenguajes permiten describir tareas orientadas a problemas en lugar de orientarlos a la computadora. Cada declaración en un lenguaje de alto nivel realiza una función reconocible, y generalmente corresponde a múltiples instrucciones en lenguaje ensamblador. Un programa llamado *compilador* traduce el programa fuente del lenguaje de alto nivel a código objeto o instrucciones en lenguaje máquina (Knaggs y Welsh, 2019, p. 6).

Como se describe en las definiciones anteriores, los lenguajes de alto nivel se caracterizan por ser independientes del hardware en el cual se ejecutan y la estructura de estos es permeable, aspecto que permite la aplicación de distintos paradigmas de programación.

Los lenguajes de alto nivel se dividen en dos grupos: lenguajes de propósito general (GPL) y lenguajes de dominio específico (DSL).

Los DSL son lenguajes diseñados para resolver problemas en un conjunto limitado de dominios, opuestamente a los lenguajes GPL, los cuales están enfocados en proporcionar soluciones de software a cualquier clase de problema. De esta manera, los DSL pueden focalizar en proveer mayor expresividad en el contexto que están definidos, permitiendo que puedan manejar mejor la complejidad dentro del dominio (Cárdenas Varón, 2009).

En la tabla I se proporciona una lista de los lenguajes de alto nivel de acuerdo con su fecha de creación.

Tabla 1

Diferentes lenguajes de programación de alto nivel con su año de creación.

Lenguaje de alto nivel	Año de creación
Fortran	1954
Lisp	1958
COBOL	1959
Simula	1964
Basic	1964
Smalltalk	1969
Prolog	1970
Pascal	1970
C	1971
Ada	1979
C++	1983
Perl	1987
Python	1991
Java	1995
C#	2000
VB.NET	2001

Nota: Recuperado de “Performance Comparison of Most Common High-Level Programming Languages” Parveen y Fatima (2016)

2.1.2 Lenguajes de programación visuales.

Los lenguajes de programación visual (LPV) se basan en el uso de expresiones visuales durante el proceso de programación tales como elementos de software manipulables interactivamente, animaciones, gráficos e iconos (entre otros). Según Kuhail (2021, como se citó en Repenning, 2017) ¨los lenguajes de programación visual se describen mediante construcciones y reglas de programación que se representan visualmente. ¨ (p.3).

Dichas expresiones visuales buscan facilitar la solución de un problema y comúnmente son utilizadas al formar la sintaxis de dicha solución, sin tener un lenguaje textual de por medio para la creación de software.

La programación visual se refiere a enfoques y métodos que utilizan elementos gráficos bidimensionales para permitir que los usuarios finales no programadores creen, amplíen y personalicen aplicaciones de software. Por lo tanto, un lenguaje de programación visual se refiere a aquel que permite el desarrollo de software mediante representaciones visuales en busca de mejorar la comprensión y simplificar el problema para los usuarios (Kuhail, 2021, como se citó en Jost et al., 2014).

2.1.2.1 Clasificación de lenguajes visuales

Los LPV utilizan representaciones visuales tales como gráficos, iconos, dibujos, animaciones y bloques, entre otros. A través de estos elementos se manipula la información visual y se da soporte a las interacciones, junto con técnicas visuales que consiguen expresar relaciones o bien transformar la información, buscando que el usuario optimice la comprensión del programa y facilite la programación en sí.

De igual modo, cada una de las representaciones visuales que emplee el LPV podrá tener características particulares, y lo mismo ocurre con las técnicas visuales que se utilicen para trabajar dichas representaciones visuales. Por lo tanto, es preciso clasificar los LPV por su tipo y particularidades. Kuhail (2021) en su investigación deduce las siguientes categorías:

Los lenguajes basados en bloques permiten a los usuarios arrastrar y soltar "bloques" (elementos de programa) desde una lista predefinida de comandos al área de desarrollo. Estos bloques se agrupan para desarrollar un programa. Este paradigma evita errores de sintaxis, lo que reduce la carga mental de los usuarios finales y les permite centrarse en los conceptos en lugar de la implementación.

Los lenguajes basados en íconos o icónicos capitalizan el uso de íconos, símbolos gráficos que representan objetos o acciones. Los iconos se pueden clasificar en iconos complejos y elementales. Los íconos elementales representan objetos (p. ej., archivo) o acciones (p. ej., eliminar, editar), mientras que los íconos complejos son íconos de objetos compuestos y oraciones visuales.

Los lenguajes de programación visual basados en formularios permiten a los desarrolladores de usuarios finales configurar un formulario en el que se agregan disparadores y acciones mediante menús desplegables de texto, o arrastrar y soltar visualmente. Algunos enfoques basados en formularios son en su mayoría visuales, mientras que otros se valen de especificaciones textuales.

Los lenguajes de programación visual basados en diagramas, también conocidos como lenguajes de diagramas o de flujo de datos, se caracterizan por conectar objetos gráficos (p. ej., cajas) mediante flechas, líneas o arcos que representan relaciones. Para comprender un programa

basado en diagramas, los usuarios recorren el diagrama. Tal diagrama utiliza diferentes medios de codificación perceptiva para representar el flujo del programa. (p.3)

2.2 Entornos integrados de desarrollo: definiciones, componentes y características.

Los entornos integrados de desarrollo, también denominados IDE por sus siglas en inglés (Integrated Development Environment) son herramientas utilizadas para el desarrollo de software, juegos o para cualquier fin que requiera codificación. Si bien es posible editar el código fuente mediante cualquier editor de texto y luego realizar la compilación en línea de comandos una opción comúnmente elegida por muchos programadores, es utilizar una herramienta que colabore con todo el proceso de programación.

Los IDE proporcionan un entorno de programación mediante el cual se puede ejecutar todo el ciclo de desarrollo de software de inicio a fin, y para llevarlo a cabo estos entornos proporcionan un marco de trabajo amigable junto con herramientas que aumentan la productividad de los desarrolladores, brindando componentes necesarios para la creación y el diseño aplicaciones.

Un entorno de desarrollo integrado es un tipo de editor mejorado para un lenguaje de programación que incorpora un compilador, pantalla de edición y herramientas de seguimiento o depuración, lo que facilita el desarrollo del programa (S.K. Satav et al. 2011)

Los entornos integrados de desarrollo pueden funcionar como aplicativos únicos, ser parte de aplicaciones existentes o bien pueden permitir la inclusión de componentes externos a fin de expandir las funcionalidades de estos. Cabe mencionar que hay IDEs que son exclusivos para un solo lenguaje de programación y hay otros que soportan tanto varios lenguajes como también el intercambio entre ellos según la finalidad del software.

Los IDE permiten que los desarrolladores comiencen a programar aplicaciones nuevas con rapidez ya que no necesitan establecer ni integrar manualmente varias herramientas como parte del proceso de configuración. Tampoco es necesario que empleen horas aprendiendo a utilizar diferentes herramientas por separado ya que todas están representadas en la misma área de trabajo (Red Hat, 2018).

Dentro de la variedad de IDE que existen, algunos están enfocados específicamente en lenguajes de desarrollo, obteniendo de esta manera características próximas a lo que denotan la programación y sus paradigmas, según el lenguaje utilizado.

A su vez, existen otros IDE que centran su foco en un propósito específico; por ejemplo, la educación en programación, donde el objetivo es enseñar utilizando los entornos integrados de desarrollo como herramienta que sustente este fin.

Según el proveedor líder de soluciones de código abierto Red Hat (Red Hat, 2018), existen una serie de características importantes que diferencian a los IDE:

- **Cantidad de lenguajes compatibles**: algunos IDE son compatibles con un solo lenguaje, por lo que resultan mejores para un modelo de programación específico. Por ejemplo, IntelliJ es conocido principalmente como un IDE de Java. Otros IDE admiten una gran variedad de lenguajes de manera conjunta, como el IDE de Eclipse, que es compatible con Java, XML, Python, entre otros.
- **Sistemas operativos compatibles**: el sistema operativo de un desarrollador determinará qué tipos de IDE son viables (salvo que el IDE esté en la nube). Además, si la aplicación en desarrollo está destinada a un usuario final con un sistema operativo específico (como Android o iOS), esto condicionará aún más el IDE que se utilizará.

- **Características de la automatización**: si bien la mayoría de los IDE incluye tres funciones esenciales (el editor de texto, la automatización de las compilaciones y el depurador), muchos admiten funciones adicionales, como la reestructuración de las aplicaciones, la búsqueda de código y las herramientas de integración e implementación continuas.
- **Impacto en el rendimiento del sistema**: si el desarrollador desea ejecutar al mismo tiempo varias aplicaciones que consumen mucha memoria, deberá considerar cuánta requiere el IDE.
- **Plugins y extensiones**: algunos IDE permiten personalizar los flujos de trabajo, de manera que se adapten a las necesidades y preferencias del desarrollador.

2.2.1 Características principales.

En general, existen una serie de características significativas que catalogan y a su vez diferencian a los IDE entre sí.

Según lo planteado por S. K. Satav et al. (2011) se destaca que entre las principales funciones de un IDE el conocimiento del código fuente, donde se denota la capacidad de un IDE para conocer las palabras clave y funciones de un lenguaje. El IDE puede utilizar este conocimiento para tareas como resaltar errores tipográficos, sugerir una lista de funciones disponibles según la situación apropiada u ofrecer la definición de una función y usar distintos colores para palabras clave y otros usos.

Otra función es la gestión de recursos de un IDE. Esta permite que, al crear aplicaciones, los lenguajes no dependan de determinados recursos tales como librerías o archivos que se

encuentran en un directorio específico. El IDE administra estos recursos y conoce las herramientas necesarias para que los errores se puedan detectar en la etapa de desarrollo, o en las etapas de compilación y construcción.

Las herramientas de depuración permiten al usuario probar minuciosamente su aplicación antes de su lanzamiento. El IDE puede indicar valores variables en ciertos puntos, conectarse a distintos repositorios de datos o aceptar diferentes parámetros de tiempo de ejecución.

Por último, cabe mencionar las funciones que se encargan de compilar y construir. Para los lenguajes que requieren una etapa de compilación o construcción, el IDE traduce el código del lenguaje de alto nivel al código máquina de la plataforma destino.

2.2.2 Entornos integrados de desarrollo para la educación.

En la última década los IDE para la programación educativa comenzaron a cobrar notoriedad dado que se incrementaron su uso, opciones y aceptación resultando que sea considerado como tema de sumo interés y foco de numerosos análisis que buscan estudiar en profundidad su impacto en el aprendizaje.

Dada esta diversidad de opciones, es natural que existan varias maneras de enseñar programación, y dichas formas son llamadas *paradigmas de programación*. Un lenguaje de programación puede evidenciar conceptos de uno o más paradigmas y ocultar conceptos a los demás permitiendo que el alumno tome decisiones o se identifique más con uno u otro (Bastos, 2015).

De todas maneras, este no siempre fue el caso ya que, si bien los sistemas educativos existen desde hace tiempo, comenzaron a surgir nuevos sistemas a partir de principios de siglo,

evolucionando y presentando nuevas variantes año tras año acorde a la evolución tecnológica. En un principio dichas herramientas de software surgieron para acompañar a los estudiantes en sus primeros pasos en la programación. Consistían en compiladores o bibliotecas específicas en lugar de entornos de programación completos que faciliten el aprendizaje y la enseñanza.

A causa de lo mencionado anteriormente, los sistemas educativos comenzaron a incorporar en sus currículos oficiales contenidos y conocimientos relacionados con la tecnología. Desde esta perspectiva podemos observar el surgimiento de una nueva metodología basada en el pensamiento computacional, la programación y la robótica (Tejera-Martínez et al., 2019 como se citó Valverde et al., 2015).

En relación con la idea anterior, la programación educativa involucra aspectos relevantes para la resolución de problemas, tales como decisiones sobre la naturaleza del problema, elección del correcto procedimiento a fin de encontrar una resolución y estrategias que permitan abordar soluciones.

Estos aspectos deben desarrollarse desde edades tempranas de manera que los estudiantes adquieran el hábito de resolver distintos tipos de problemas de manera sistemática, aplicando o no el uso de una computadora para su solución, aunque es una gran alternativa si se busca desarrollar habilidades en el marco del pensamiento computacional. Este se define como un proceso de pensamiento a través de habilidades que resultan fundamentales en programación (habilidades CT), para resolver problemas independientemente de la disciplina (L. Zhang, J. Nouri, 2019).

Katz (2016) señala la necesidad de fomentar el desarrollo de estas competencias junto con el pensamiento computacional y la programación. En la tabla I se observan dichas competencias, listadas en categorías, junto a las capacidades y habilidades que desarrollan.

Tabla 2.

Competencias clave

Competencia	**Habilidad / Capacidad**
Categoría 1. Usar las herramientas de forma interactiva	
Uso interactivo del lenguaje, símbolos y textos	Destrezas lingüísticas orales y escritas, de computación y matemáticas.
Uso interactivo del conocimiento y la información	Reconocer y determinar lo que no se sabe; identificar, ubicar y acceder a fuentes apropiadas de información; evaluar la calidad, propiedad y el valor de dicha información, así como sus fuentes y organizar el conocimiento y la información.
Uso interactivo de la tecnología	Relacionar las posibilidades que yacen en las herramientas tecnológicas con sus propias circunstancias y metas e incorporar la tecnología en prácticas comunes
Categoría 2. Interactuar en grupos heterogéneos	
Relacionarse bien con otros	Fomentar la inteligencia emocional, respetar y apreciar los valores, las creencias, culturas e historias de otros, desarrollar la empatía, la autorreflexión, el manejo efectivo de las emociones, el conocimiento sobre uno mismo y los demás, etc.
Cooperar y trabajar en equipo	Habilidad de presentar ideas y escuchar las de otros, entendimiento en dinámicas de debate y seguimiento de una agenda, construcción de alianzas, negociar, tomar decisiones, etc.
Manejar y resolver conflictos	Analizar los elementos e intereses en juego, los orígenes del conflicto y el razonamiento de todas las partes, reconociendo que hay diferentes posiciones posibles; identificar áreas de acuerdo y desacuerdo; recontextualizar el problema y priorizar las necesidades.
Categoría 3. Actuar de manera autónoma	
Actuar dentro del contexto del gran panorama	Comprensión de patrones, conocimiento del sistema en el que existen (estructuras, cultura, prácticas, reglas, roles y normas sociales), identificación de consecuencias de sus acciones y reflexión de las mismas, tanto de forma individual como colectiva.
Formar y conducir planes de vida y proyectos personales	Definición de proyecto y metas, identificación, balance y evaluación de recursos, priorización de metas, aprender de acciones pasadas

	proyectando resultados futuros y monitorear el progreso haciendo los ajustes necesarios.
Defender y asegurar derechos, intereses, límites y necesidades	Comprender intereses propios y colectivos, conocer las reglas y principios para basar un caso, construir argumentos para que los derechos y necesidades propios sean reconocidos, sugerir arreglos o soluciones alternativas.

Nota: "Lenguajes de programación y desarrollo de competencias clave.", Tejera-Martínez et al., (2019).

En función de lo planteado, desde la perspectiva de Zhang y Nouri (2019) en la actualidad existen 3 categorías de definiciones presentes para las habilidades de pensamiento computacional, que describimos a continuación:

Las *definiciones genéricas* se centran en el proceso de pensamiento para la resolución de problemas que conduce a una solución de pasos computacionales o algoritmo. Por lo tanto, las habilidades de CT que subyacen a esta definición pueden volverse algo generales como "un conjunto de habilidades de aplicación universal" de lectura, escritura y aritmética.

Las *definiciones operativas* y sus correspondientes habilidades de CT se pueden caracterizar como formular, organizar, analizar, automatizar, presentar, implementar y transferir.

Las *definiciones educativas,* donde las habilidades de pensamiento computacional según el marco de trabajo utilizado serán la abstracción, algoritmos, descomposición, datos, paralelización, pruebas y depuración; también estructuras de control, generalización y eventos, entre las más relevantes.

En consecuencia, con estas definiciones surge la necesidad de poder introducir a los alumnos a la programación mediante entornos de desarrollo que resulten sencillos de utilizar y que, a su vez, permitan realizar procedimientos que incluyan estructuras básicas de control, decisión y repetición. De esta manera, en los ciclos superiores se podrá conducir a los estudiantes a desarrollar habilidades que se puedan utilizar en tipos de entornos más demandantes y se los

prepare para utilizar lenguajes textuales profesionales bajo el paradigma de objetos o cualquier otro.

A raíz de la problemática expuesta, y en adición a la necesidad de incluir la programación dentro del sistema educativo, es que surgen los Entornos Integrados de Desarrollo para la educación como una herramienta introductoria a la programación.

Desde el punto de vista de Algaraibeh (2020), estos IDE están diseñados para utilizarse en los cursos de introducción a la programación donde se enfocan en conceptos básicos de programación y ofrecen comentarios informativos. Algunos IDE educativos especializados priorizan el soporte para el primer enfoque de objeto que se dedica a reconocer los conceptos orientados a objetos: clase, objeto y miembros. Por el contrario, otros se centran en la retroalimentación personalizada y el aprendizaje de conceptos básicos: variables, tipo de datos, operación matemática y lógica, expresión, estructura de iteración, condicional y función. A su vez, también existen entornos que utilizan objetos gráficos basados en bloques y se encuentran destinados a estudiantes más jóvenes. Esos entornos son diseñados para cubrir un pequeño subconjunto de conceptos de programación. Además, la utilización de objetos gráficos que representan construcciones de programas que se insertan y mueven dentro de un entorno gráfico resulta compleja debido al alto nivel de abstracción. Dichos entornos de programación visual son más fáciles de usar que los entornos basados en texto, pero pueden dificultar el aprendizaje de los conceptos de programación más profundos.

Capítulo 3 – Transición entre lenguajes de programación visuales y textuales.

El tercer capítulo se aborda la problemática de la investigación: la transición entre lenguajes textuales y visuales. Se presentan estrategias y técnicas existentes, tanto tecnológicas como educativas.

2.1 La importancia de la transición entre lenguajes de programación.

En la actualidad, las personas coexistimos en un mundo donde las tecnologías forman parte prácticamente de todo, agilizando y optimizando las actividades realizadas en cualquier contexto donde se estén utilizando. Es por ese motivo que el rol de la programación dentro de la tecnología se vuelve fundamental para acrecentar su avance. En este contexto, las prácticas y conceptos relacionados al área de conocimiento algoritmos y programación reciben una atención creciente en el campo educativo (Rodriguez et. al 2019).

Como observamos en el capítulo anterior, la programación posee distintos enfoques: pueden utilizarse lenguajes visuales o textuales, y no existen tanto una forma de practicarla como una única herramienta donde desarrollarla. Es por ello que uno de los principales retos de la sociedad actual radica en la enseñanza de la programación como asignatura básica. En muchos países se promueve el desarrollo de propuestas curriculares que adopten prácticas y conceptos fundamentales sobre Ciencias de la Computación como contenidos escolares para la educación secundaria (Rodriguez et. al 2019).

De todas formas, dicha inclusión debe poder efectuarse tanto en los ciclos iniciales como en los ciclos superiores de educación, aprovechando las cualidades positivas de cada paradigma de programación, sea textual o visual, y así lograr que el aprendizaje de los alumnos se vuelva

efectivo, independientemente de la etapa del ciclo educativo para que a futuro los estudiantes puedan desempeñarse como profesionales del campo.

En los últimos años se realizaron numerosas investigaciones que buscan definir enfoques y construir entornos destinados específicamente a la enseñanza y el aprendizaje de la programación en ambientes formales y no formales (Rodriguez et. al 2019)

Sin embargo, existen desafíos emergentes para la enseñanza de las ciencias de la computación al momento de plantear una incorporación sostenida, rigurosa y progresiva a lo largo de los años de escolaridad.

Se espera que en el transcurso del ciclo escolar los estudiantes logren construir habilidades para desarrollar programas utilizando lenguajes de programación convencionales basados en texto. El paso a un entorno de programación tradicional suele resultar un proceso frustrante para la mayoría de los estudiantes.

De acuerdo con Deshmukh (2018), muchas herramientas de programación visual han permitido el desarrollo de un currículo exitoso de ciencias de la computación y cursos interdisciplinarios para promover el pensamiento computacional. Las herramientas proporcionan una interfaz visual para la programación a través de bloques de código listos, animaciones, escenas, personajes y otros. Por lo tanto, reducen la frustración de los nuevos estudiantes frente a la sintaxis de la programación.

En síntesis, y de acuerdo con (Rodriguez et. al 2019), resulta de suma importancia trabajar el problema de la transición entre distintos paradigmas de programación a partir investigaciones y del desarrollo de entornos que contemplen la problemática y que, a su vez, se constituyan en soporte a prácticas pedagógicas vinculadas con las necesidades de formación para evitar que los estudiantes se frustren al dar este paso.

2.2 Razones por las que se producen discordancias entre lenguajes de programación.

2.2.1 Problemas en la enseñanza del pensamiento computacional.

La computación y la enseñanza de ella integran un campo de investigación que posee una vasta literatura y que, a su vez, se encuentra motivada por los continuos avances tecnológicos que la acompañan.

El Departamento de Educación del Reino Unido (2015) describe la computación de la siguiente manera: “La computación es un concepto aún más grande. El núcleo de la informática es la ciencia de la computación, en la que se enseña a los alumnos cómo utilizar este conocimiento a través de la programación.”

Por otra parte, a medida que los sistemas educativos evolucionan, también lo hace la enseñanza de la informática. Esto condujo a que, en la actualidad, los sistemas educativos adopten lo que se considera *pensamiento computacional* (CT).

Zhang (2019) sostiene que el pensamiento computacional es una habilidad del siglo XXI que las generaciones futuras deben desarrollar. Afirma también que esta creencia ha sido reconocida internacionalmente y un número creciente de sistemas educativos han integrado el CT en su educación obligatoria en los últimos años, basándose en una revisión realizada en el año 2014 donde se presentaron los currículos educativos actualizados país por país.

De manera que el soporte tecnológico a la enseñanza del pensamiento computacional como habilidad para resolver problemas a través de entornos adecuados para el aprendizaje resulta una pieza fundamental durante el proceso de enseñanza.

El concepto de CT se introdujo originalmente en el libro de Seymour Papert "Mindstorms: Children, Computers, and Powerful Ideas" (1980) con su creación del lenguaje de programación LOGO. Papert se refirió a CT principalmente como la relación entre la programación y las habilidades de pensamiento. Él creía que las construcciones de los estudiantes a través de la programación con LOGO podrían facilitar su pensamiento procedimental en múltiples disciplinas. (Zhang, 2019)

Posteriormente, el término *pensamiento computacional* fue redefinido por Wing (2006), donde el autor argumenta que se trata de una habilidad crítica para todas las personas (no solo para los graduados en el campo de la informática) e implica resolver problemas, diseñar sistemas y comprender el comportamiento humano basándose en los conceptos fundamentales de las ciencias de la computación. (p. 33)

En relación con estos conceptos, dentro de la informática encontramos distintas maneras de programar (como hemos mencionado anteriormente) y para ellos existen diversos enfoques, lo que plantea una situación que conduce al programador a que sea capaz de adaptarse a una variedad de contextos y también a perfeccionar las formas de programar. Dada esta multiplicidad de opciones, es natural que existan distintas maneras de enseñar programación, sin una definición universal, permitiendo que el alumno tome decisiones o se identifique más con un lenguaje u otro. No obstante, profesionalmente estos lenguajes ofrecen diferencias que deben ser entendidas y bien exploradas para su uso adecuado en el futuro.

Por consiguiente, la falta de una definición universal ha causado problemas en la educación en CT. La definición de CT aún no es una constante fija. Por ejemplo, en 2010 Wing actualizó su definición anterior de CT de 2006. Los profesores e investigadores han utilizado herramientas de programación para entrenar habilidades distintas de las consideradas habilidades de CT (tabla 2).

Como señalan Selby y Woollard (2013), el diseño de un currículo consistente y una evaluación adecuada se encuentran fuera de discusión sin saber lo que implica CT.

Por lo tanto, es importante que estas diferencias sean percibidas por los estudiantes desde la educación básica para que en el futuro profesional logren desenvolverse correctamente sin importar cuál sea el paradigma que utilicen.

2.2.2 Problemas de sintaxis

En un lenguaje de programación la sintaxis es con frecuencia el principal desafío para el aprendizaje de los estudiantes. Según Denny (2011) en su obra *Comprender la barrera sintáctica para principiantes* detectó que incluso los alumnos con mayor recorrido en la programación tenían problemas de sintaxis en más del 50% de los casos estudiados, concluyendo que la sintaxis es un gran problema en las primeras etapas de aprendizaje.

La complejidad radica en las condiciones sintácticas que precisan los lenguajes basados en texto para su funcionamiento. Como expresa Van Syl et.al (2016, como se citó en Kerman, 2002) si el código no se escribe exactamente de acuerdo con las reglas sintácticas del lenguaje de programación, no se puede ejecutar el programa y se muestran mensajes de error. Como resultado, los programadores están constantemente ocupados con el manejo de errores. Esta puede ser una tarea frustrante e implica cierto tiempo que los programadores novatos entiendan los mensajes y puedan corregir los errores (Van Syl, 2016, como se citó en Gomes & Mendes, 2007).

Desde esta perspectiva Cheung et.al, (2009) afirma que el problema más importante con los IDE tradicionales radica en que el lenguaje de programación y su sintaxis se convierten en el foco del aprendizaje, en lugar de focalizar en las habilidades para resolver inconvenientes. Los estudiantes a menudo terminan concentrándose en problemas sintácticos en lugar de problemas

lógicos. Por lo tanto, la fortaleza de las habilidades de resolución de problemas de los estudiantes depende de su desarrollo personal en lugar de la capacitación informática que reciben en su curso.

Por otra parte, Kölling (2010) manifiesta que los estudiantes de la escuela primaria aún no son lo suficientemente maduros para tratar la sintaxis en un lenguaje de programación, por lo que deben comenzar el aprendizaje en un entorno de programación que elimine el problema sintáctico, permitiendo que el estudiante se enfoque en resolver el problema propuesto.

Por lo tanto, notamos que el conocimiento de las reglas sintácticas de un lenguaje de programación textual resulta indispensable para que los estudiantes puedan ejecutar la sintaxis del lenguaje correctamente y así lograr programas válidos. Del mismo modo, se evidencia que los problemas de sintaxis son un inconveniente común al que se enfrentan los programadores novatos cuando comienzan a programar en un lenguaje de programación basado en texto.

2.3 Estrategias y herramientas para abordar la transición entre lenguajes de programación.

Existen varios frentes de investigación en busca de cómo y por dónde iniciar la transición entre los paradigmas de programación textuales y visuales. Por ese motivo es que a continuación se plantean las estrategias y herramientas que intentan abordar la problemática.

2.3.1 Herramientas para abordar la transición

A modo de comienzo es importante destacar la importancia del aprendizaje de conceptos generales de programación. El autor Algaraibeh (2020) los describe como *icebergs* que esconden muchos detalles. Por un lado, los investigadores de la psicología de la programación instan al

diseñador del lenguaje de programación a emplear principios cognitivos para reducir las barreras de la programación (Algaraibeh 2020, según se citó en Green). Por otro lado, resulta muy importante que el alumno comprenda cómo se fusionan ciertos temas relacionados a las ciencias de la computación tales como el pensamiento computacional, la resolución de problemas, los lenguajes de programación, los sistemas operativos de computadora, los software de sistemas y las matemáticas.

En conclusión, estas herramientas idealmente deberían orientar al diseño y construcción de ambientes destinados a facilitar los procesos de transición de manera tal que se posibilite trabajar con un lenguaje de representación visual, visualizando el código producido automáticamente, y viceversa (Rodriguez et. al 2019).

En este sentido, Weintrop (2015) afirma que idealmente se busca sostener las ventajas de la representación gráfica y la característica de mapeo directo de las operaciones sobre el escenario, con intención de posibilitar una retroalimentación ágil y la detección y corrección de errores. Al mismo tiempo que se mantiene la tarea en el plano de lo concreto se proporciona la flexibilidad, versatilidad y legibilidad del texto.

Idealmente estas herramientas deberían contener las siguientes características:

Tabla 3

Características ideales de las herramientas para soportar la problemática.

Característica	**Descripción**
Multilenguaje	La herramienta debería proveer la posibilidad de generar e interactuar con más de un lenguaje de programación. Preferentemente, que algunos de ellos sean los mismos que luego serán utilizados en ambientes solo texto

	para continuar con la profundización de la disciplina.
Modalidad dual	Debería ser posible generar código a partir de bloques, y viceversa, en tiempo real. Las construcciones obtenidas a partir de los bloques, y viceversa, deben ser lo más directas posibles, para que la equivalencia entre ambas estructuras sea clara y no haya dudas de que representan al mismo algoritmo de alto nivel.
Autoevaluativa	No debería ser necesaria la ayuda de un docente para que el estudiante sepa si su algoritmo resolvió el problema o no. De todas maneras, esto no excluye una instancia de evaluación posterior de eficiencia, elegancia y posibles alternativas de solución diferentes.
Instalable	Una herramienta puramente web no fácilmente instalable de manera local no estaría al alcance de toda la comunidad educativa.
Conjunto reducido del lenguaje	Especialmente si uno o más de los lenguajes de traducción es de uso comercial como Java, Python, JS, etc. Es importante acotar el conjunto de instrucciones para no dispersar con las mismas el objetivo del aprendizaje siendo las instrucciones incluidas posibles de incorporar en etapas posteriores con otros ambientes
Mapeo directo	Las operaciones programadas se mapean directamente sobre el mundo ofreciendo una retroalimentación instantánea de la actividad en la pantalla o el entorno físico.
IDE simple	IDEs simples que integren algunas funciones de alerta visual de errores, reconocimiento de sintaxis y posibilidad de ejecutar el código

Nota: *"Transición desde programación basada en bloques a basada en texto: una revisión del campo* Rodriguez et al., (2019).

2.3.2 Estrategias para abordar la transición entre lenguajes visuales y textuales.

2.3.2.1 Estrategia basada en lenguajes de desarrollo híbridos

Los lenguajes de desarrollo híbridos poseen una interfaz que les permite visualizar tanto elementos visuales como textuales al mismo tiempo, y también permiten la combinación de un lenguaje textual en una interfaz basada en bloques de desarrollo Weintrop (2015).

Sin duda, los enfoques visuales y textuales revisten beneficios, por lo que combinarlos podría tener el mejor resultado posible en jóvenes estudiantes.

El primer estudio encontrado sobre el concepto de lenguajes de desarrollo híbridos fue realizado por Cilliers et al. (2005) Ambos autores querían examinar qué efecto tendría la integración de una notación icónica en un entorno de desarrollo textual. En ella identificaron que las notaciones de programación visual ofrecen beneficios sobre las notaciones de programación textuales, a tiempo que reconocieron que los LPV no eran una solución independiente.

Koitz y Slany (2014) también realizaron una investigación similar al comparar un LPV y un entorno de desarrollo para móviles que utiliza una combinación de programación textual y elementos visuales. Luego de realizar cuatro pruebas en distintos idiomas, los resultados mostraron que el enfoque híbrido era más beneficioso que el enfoque puramente visual.

En las últimas décadas, ha cobrado una gran relevancia la cantidad de investigaciones sobre lenguajes de desarrollo híbridos que utilizan lenguajes textuales existentes, que a su vez se combinan con un LPV basado en bloques de desarrollo.

En 2011, Federici (2011) combinó un lenguaje de programación estructurado, permitiendo la creación de bloques y funciones personalizadas dentro del IDE en un sistema de bloques.

Matsuzawa et al. (2015), proporcionan en su investigación un ejemplo de un lenguaje de bloques basado en Java. Con su estudio, la herramienta permitió la traducción directa entre bloques y el lenguaje Java basado en texto. El autor planteó que los estudiantes deberían comenzar con este enfoque de bloques y avanzar gradualmente hacia un entorno totalmente textual.

De este modo se puede observar que los lenguajes de desarrollo híbrido representan una estrategia creciente debido a la dualidad de su naturaleza al combinar lenguajes textuales y lenguajes visuales, y a fin de minimizar la brecha entre ambos paradigmas.

2.3.2.2 Estrategia basada en bloques

La enseñanza de un LPV basado en bloques tiene un gran impacto entre los más jóvenes, es decir, en los primeros años de escolaridad.

Sandoval-Reyes et al. (2011) realizaron un análisis de tres entornos principales de programación de bloques: Scratch, Alice y App Inventor, mientras también observaban a Greenfoot. Presentaron la idea de que este tipo de entorno provee una pedagogía sólida debido a que vincula a los usuarios con sus intereses, asignando directamente las ideas a las instrucciones en la interfaz y ocultando complejidades innecesarias al usuario novato. El enfoque de bloques puede funcionar en todo tipo de entornos, como lo demuestra Slany (2012) en su trabajo, permitiendo a los usuarios desarrollar programas directamente en su teléfono, e incluso pueden desarrollar controladores para otros dispositivos relacionados con la robótica educativa como Lego Mindstorms. Para muchos estudiantes novatos que se encuentran introduciéndose en la programación esta se vuelve una perspectiva realmente emocionante.

2.3.2.3 Estrategia basada en juegos

Los LPV que se utilizan para la enseñanza de programación de una manera lúdica resultan muy similares al enfoque basado en bloques, aunque tienen un objetivo diferente. Como lo expresa Cooper (2010) su idea era proporcionar a los estudiantes una experiencia seria de programación anterior a la educación secundaria a través del IDE Alice. Este IDE permite a los usuarios construir un mundo con animaciones, siendo también funcional, utilizando bloques de código que se arrastran y sueltan. También contiene un editor que permite a los usuarios diseñar un conjunto de objetos en el espacio 3D. Todo el código subyacente todavía se trata con la función de arrastrar y soltar después de esta configuración visual inicial. Cooper (2010) señaló que, a diferencia de la animación basada en algoritmos, los sistemas de visualización de programas permiten al alumno crear sus propias animaciones. Una de las mayores diferencias entre Alice y los LPV tradicionales de bloques radica en que admite el enfoque de programación orientado a objetos, aunque de forma limitada, cumpliendo con la dualidad entre lenguajes textuales y LPV.

Capítulo 4 – Planificación.

En el cuarto capítulo se desarrolla la primera etapa de las revisiones sistemáticas, la Planificación. Se realiza el desarrollo del protocolo de la investigación donde se detalla la pregunta que dirige la búsqueda, y se muestra la estrategia que se sigue con los criterios de inclusión y exclusión. Por último, presentamos cómo se van a tratar los resultados obtenidos.

4.1 Identificar la necesidad de revisión.

En los últimos años y hasta la actualidad, de la mano de la evolución tecnológica constante, la programación se ha vuelto una habilidad cada vez más necesaria y una asignatura cada vez más común dentro de los planes escolares.

Es por lo que surge la necesidad de reunir evidencia empírica respecto a qué estrategias, métodos y herramientas existen para que los estudiantes puedan lograr el aprendizaje.

Cabe destacar que este punto es complementario al desarrollado en el capítulo 1.

4.2 Definir las preguntas de investigación que conducirán la búsqueda

La pregunta de investigación principal, según se expresó en el capítulo 1, es la siguiente:

> "¿Cuál es la estrategia educativa más utilizada para realizar la transición entre lenguajes de programación visuales y textuales surgida en los últimos 30 años?"

Para responder a la misma y conducir la metodología de la revisión sistemática, se elaboraron sub preguntas de revisión que se exponen a continuación en la tabla 4.

Tabla 4

Sub preguntas de búsqueda

Sub pregunta de investigación	**Objetivo**
¿Propone el estudio herramientas de software que permitan realizar la transición de un lenguaje visual a un lenguaje textual?	Determinar, por sí o por no, si encontramos en el trabajo alguna herramienta de software que contemple la transición entre lenguajes visuales y textuales.
• SI	

¿Qué herramienta de software propone el trabajo?	Determinar qué tipo de herramienta de software propone el trabajo: IDE, Framework, librería, lenguaje, robots, etc.
¿Propone el trabajo una estrategia educativa para acompañar la transición de un lenguaje visual a uno textual?	Determinar la estrategia educativa aplicada durante el aprendizaje o la transición de un lenguaje de programación visual a uno textual o viceversa.
¿Qué efecto o efectos reporta el estudio?	Verificar cuál es el efecto relacionado a la problemática que se reporta en el trabajo.
¿Qué nivel del ciclo educativo se involucra en el estudio?	Identificar la etapa del ciclo educativo que abarca el estudio.
• NO	
¿El trabajo presenta resultados de algún experimento donde se ensaye la transición de un lenguaje visual a uno basado en texto?	Determinar si el trabajo presenta un estudio experimental de la problemática e incluye resultados de prueba.
¿Propone una estrategia educativa o alternativa para ejecutar la transición de un lenguaje visual a uno textual?	Determinar la estrategia educativa aplicada durante el aprendizaje o la transición de un lenguaje de programación visual a uno textual o viceversa.
¿Qué efectos analiza o reporta el estudio?	Verificar cuál es el efecto relacionado a la problemática que se reporta en el trabajo.
¿Qué nivel del ciclo educativo se involucra en el estudio?	Identificar la etapa del ciclo educativo que abarca el estudio.

4.3 Desarrollo protocolo de revisión

El protocolo de revisión enumera los pasos que se realizaran para llevar a cabo una revisión sistemática de la literatura. La definición de dicho protocolo es necesaria para reducir la posibilidad de sesgo.

4.3.1 Los términos de búsqueda

El método utilizado para construir los términos de búsqueda se compuso de 4 pasos:

El primer paso es la identificación de los términos principales considerando la pregunta de investigación (ver Tabla 5)

Tabla 5

Identificación de términos principales de búsqueda.

Población	SLR
Intervención	Transición de lenguajes de desarrollo visuales a lenguajes basados en texto.
Resultados	Herramientas, técnicas, estrategias, metodologías que respalden la transición de lenguajes de desarrollo visuales a lenguajes basados en texto.
Contexto	Educativo

El segundo paso es la identificación de sinónimos, palabras alternativas o abreviaturas de los términos principales teniendo en cuenta las palabras clave que se encuentran en los artículos sobre TVC (ver Tabla 6).

Tabla 6

Identificación de sinónimos respecto a los términos principales de búsqueda.

Término principal	**Término alternativo**
Language	programming language
Tools	integrated development environment

Programmer	"development" , "developer"
Education	"classroom" , "educational","learn","teaching", "schooling" , "schoolhouse","schoolroom"
Kids	“kids” ,"young"

El tercer paso consta de la aplicación de un OR booleano para incorporar sinónimos, palabras alternativas o abreviaturas (ver Tabla 7).

Tabla 7

Incorporación de sinónimos a los términos claves de búsqueda.

Término principal	**Término alternativo**
Language	programming language
Tools	integrated development environment
Programmer	"development" OR "developer"
Education	"classroom" OR "educational" OR "learn" OR "teaching" OR "schooling" OR "schoolhouse" OR "schoolroom"
Kids	“kids” OR "young"

Y, por último, el cuarto paso consta del uso de enlace booleano y de los términos principales (ver Tabla 8).

Tabla 8

String de búsqueda final.

String final
("programming language" OR "integrated development environment") AND ("programmer" OR "development" OR "developer") AND ("education" OR "classroom" OR "educational" OR "learn") AND ("kids" OR "teaching" OR "schooling" OR "schoolhouse" OR "schoolroom" OR "young")

4.3.2 Estrategia de búsqueda

Es la estrategia que utilizaremos para buscar los estudios primarios incluyendo los términos de búsqueda y recursos que se desean buscar. Para llevar a cabo esta exploración es necesario desarrollar una estrategia que permita identificar los posibles estudios primarios

Basándose en el propósito de una revisión sistemática (relevar y evaluar la cantidad de evidencia existente que aborde las preguntas que impulsaron la investigación) se llevará a cabo un proceso riguroso e imparcial para obtener la mayor cobertura de trabajos y fuentes de información posibles.

4.3.3 Criterios de inclusión y exclusión

Se indican los criterios que se aplicaron en la selección de estudios para determinar cuáles son incluidos o excluidos de la revisión sistemática.

Tabla 9

Criterios de inclusión y exclusión.

Criterios inclusión	Trabajos en idioma inglés.
	Trabajos publicados entre 1980 y 2021.
	Trabajos avalados académicamente.
	Trabajos que satisfagan la cadena de búsqueda.
	Trabajos relacionados a entornos integrados de desarrollo, lenguajes de programación o proyectos de software libres y modificados

	para facilitar el desarrollo de software en el marco educacional. Trabajos que mencionan el uso de entornos integrados de desarrollo como medio educativo para facilitar la programación de aplicaciones gráficas, robótica, juegos y simulaciones Trabajos relacionados a experimentos, simulaciones y competencias tecnológicas, las cuales permitan identificar resultados de actividades que hayan utilizado entornos integrados de desarrollo experimentales, lenguajes de programación de dominio específico, proyectos de código abierto adaptados al contexto educacional, y que faciliten la programación. Trabajos que contengan información de entornos de desarrollo que presentan interfaces visuales con editor de texto.
Criterios de exclusión en el texto completo.	Trabajos que no tengan relación con el desarrollo de herramientas, lenguajes de programación o entornos integrados educacionales de desarrollo para mejorar el aprendizaje de la programación. Trabajos que nos propongan estrategias o metodologías de enseñanza a fin de facilitar la enseñanza de la programación en el ámbito educativo. Trabajos que aborden el tema de la transición de lenguajes visuales a lenguajes textuales desde un contexto distinto al educativo. Trabajos de investigaciones experimentales que no posean o emitan resultados comprobables. Trabajos de investigaciones experimentales que no estén evaluadas académicamente.

Criterios de exclusión para títulos y resumen.	Trabajos que no aborden de forma explícita o implícita la transición entre lenguajes de programación visuales y textuales. Trabajos donde se describen soluciones aplicables a un contexto distinto al educativo. Trabajos de robótica educativa desde una perspectiva programática distinta. Trabajos no relacionados con la programación desde el punto de vista educacional. Trabajos duplicados. Trabajos en un idioma distinto al inglés.

4.3.4 Procedimiento de selección de estudios

Durante este proceso se seleccionaron todos los estudios que hayan abordado la pregunta de investigación. Este procedimiento transitó necesariamente por los criterios de inclusión y exclusión de los trabajos.

Partiendo de los criterios definidos previamente, se procedió a realizar una búsqueda bibliográfica lo más amplia posible con el fin de identificar los estudios que pudieran cumplir con los criterios de selección. Esta exploración tal como fue planteado en la estrategia de búsqueda, se realizó mediante SCOPUS a fin de garantizar la formalidad de los trabajos y se dividió en tres etapas.

- En la primera etapa se aplicó la estrategia de búsqueda para identificar los posibles trabajos primarios y excluir los duplicados.
- En la segunda etapa se seleccionaron los trabajos que aplicaban, en base a su título y resumen, según los criterios de inclusión y exclusión previamente definidos.

- En la tercera etapa cada potencial trabajo primario seleccionado se examinó a texto completo para decidir sobre su inclusión o exclusión.

Una vez seleccionados los trabajos que cumplen con los criterios establecidos y son consonantes con las preguntas de investigación, fueron examinados para determinar las características metodológicas de cada uno como, por ejemplo el diseño de este; si se trata de un trabajo experimental o un metaanálisis de la temática. Dicha averiguación permitió una clasificación de estos estableciendo una relación con los resultados de la investigación,

Toda esta información se documentó por medio de la creación de una planilla en la que los trabajos primarios son las filas y las columnas son las variables que moderan y clasifican los resultados de la selección de estudios, permitiendo una descripción de sus características y la codificación de los trabajos.

4.3.5 Evaluación de calidad

Para analizar la calidad de los estudios primarios incluidos se desarrolló una lista de verificación que contiene preguntas de evaluación de calidad de trabajo.

- ¿El estudio identifica la problemática de transición de un lenguaje visual a uno textual?
- ¿El estudio contiene una propuesta para resolver el problema de algún modo?
- ¿El estudio observa efectos en el aprendizaje programático de lenguajes textuales, luego de realizar una experiencia en lenguajes visuales?
- ¿El estudio plantea una estrategia o herramienta definida para resolver el problema?

Para cada una de las preguntas, se empleó la siguiente escala de puntos a fin de categorizar la calidad de los documentos seleccionados.

Tabla 10

Puntuaciones para la evaluación de calidad.

Descripción	Puntaje
Escaso	0 a 1 puntos
Aceptable	1,1 a 2,5 puntos
Bueno	2,6 a 3,5 puntos
Muy bueno	3,6 a 4 puntos

4.3.6 Estrategia de extracción de datos

El siguiente punto hace referencia a la estrategia de extracción de datos utilizada e implica la recolección de la información más relevante de cada trabajo que fue incluido.

La información fue documentada en una planilla previamente diseñada junto al tutor, donde se incorporó toda la información detallada y pertinente, basándose en las preguntas y sub preguntas que guían nuestra investigación, junto con otros datos de interés para el análisis, síntesis e interpretación de los mismos. Los datos incluidos en nuestra planilla fueron los siguientes:

- número de trabajo
- título del trabajo
- año de publicación
- autor

- resumen
- criterios de inclusión y exclusión.
- preguntas de investigación
- observaciones adicionales

Para la obtención de la información se analizaron los trabajos resultantes de la búsqueda y los puntos mencionados anteriormente se obtuvieron, en parte, con datos nominales de cada trabajo, tales como el título, año de publicación, autor y/o resumen. Luego, fue necesario el análisis individual de cada trabajo revisando en primera instancia los criterios generales de inclusión y exclusión para luego continuar verificando los criterios de exclusión para títulos y resumen; seguido por último de la lectura de cada estudio para responder las preguntas de investigación, cómo así también realizar la evaluación de calidad.

4.3.7 Obtención de datos

Se describe a continuación cómo se clasifica la información obtenida, luego de ejecutar las preguntas de investigación. El resultado de la clasificación permitirá analizar con mayor precisión los datos obtenidos.

De forma que para responder la pregunta de investigación inicial:

“¿Cuál es la estrategia educativa más utilizada para realizar la transición entre lenguajes de programación visuales y textuales surgida en los últimos 30 años?”

Se realizó a los estudios seleccionados una serie de preguntas adicionales para poder realizar una clasificación de los trabajos.

- ¿Propone el estudio herramientas de software que permitan realizar la transición de un lenguaje visual a un lenguaje textual?
    - Si: Trabajos que proponen una herramienta para llevar a cabo la transición.
    - No: Trabajos que no proponen una herramienta para llevar a cabo la transición.

A partir de esta primera clasificación, se evaluarán los trabajos dependiendo de la respuesta de la sub pregunta 1.

Por la respuesta “Sí” se analizaron resultados distintos en relación con la respuesta “No”. Es por ello que se adicionaran sub preguntas para ambos casos.

Cuando la respuesta fue “Si” se realizaron las siguientes preguntas:

- ¿Qué herramienta de software propone el trabajo?
- ¿Propone el trabajo una estrategia educativa para acompañar la transición de un lenguaje visual a uno textual?
- ¿Qué efecto o efectos reporta el estudio?
- ¿Qué nivel del ciclo educativo se involucra en el estudio?

Cuando la respuesta fue “No” se realizaron las siguientes preguntas.

- ¿El trabajo presenta resultados de algún experimento donde se ensaye la transición de un lenguaje visual a uno basado en texto?
- ¿Propone una estrategia educativa o alternativa para ejecutar la transición de un lenguaje visual a uno textual?

- ¿Qué efectos analiza o reporta el estudio?
- ¿Qué nivel del ciclo educativo se involucra en el estudio?

4.4 Resultados obtenidos

El objetivo de este punto guarda relación con la forma en que se sintetizaran los datos extraídos. A partir de dicha síntesis, se decidió que los resultados serán divididos y analizados por distintos aspectos. Por un lado, se realizará un análisis cuantitativo sobre los trabajos, mientras que por otro se analizarán los trabajos cualitativamente. Si surgiese algún tipo de conclusión y análisis sobre los resultados obtenidos se distinguirán en profundidad en el capítulo correspondiente.

4.4.1 Análisis cuantitativo

Se realiza un análisis cuantitativo desde el que se podrán exponer las cantidades de estudios en relación con alguna característica definida para la búsqueda. Asimismo, cada sub pregunta que consideremos de importancia se sumará a las opciones encontradas. A modo de ejemplo podemos citar: para la sub pregunta: *¿Qué herramienta de software propone el trabajo?*, se indicarán las cantidades de IDEs, lenguajes de programación, prototipos y librerías entre otros.

4.4.2 Análisis cualitativo

Los trabajos clasificados y evaluados con el sistema de valoración de calidad se listarán en función de la medida de calidad obtenida en el análisis. Como resultado final se podrá obtener un

mapa global que muestre los trabajos más relevantes para la materia (en relación con la evaluación de calidad realizada sobre ellos) en cada uno de los temas y subtemas clasificados.

Capítulo 5 – Ejecución

En el quinto capítulo se lleva a cabo la revisión sistemática propiamente dicha. Es decir, se ejecuta el protocolo de la Revisión Sistemática de Literatura definido previamente. Para ello se procede a leer cada uno de los trabajos seleccionados y por cada uno se responden las preguntas definidas en dicho protocolo. Al mismo tiempo se completa un documento que englobe todas las respuestas que luego nos permitirán extraer conclusiones.

5.1 Ejecución de la estrategia de búsqueda.

Se ejecuta la estrategia de búsqueda y se recuperan los estudios primarios. Estos estudios son los estudios originales utilizados en la revisión sistemática de la literatura.

La estrategia de extracción de datos se basó en la información resultante tras la ejecución de los términos de búsqueda en la base de datos SCOPUS.

El número de trabajos obtenidos fue de 5611 trabajos.

Los resultados fueron exportados a una planilla para su posterior análisis, síntesis e interpretación. La exportación se realizó respetando todos los parámetros por defecto que para cada artículo se registraron los siguientes datos: Titulo, Abstract, Autor, Año, Link, Tipo de documento, Doi.

5.2 Proceso de selección de trabajos.

Durante este proceso se seleccionaron todos los estudios que hubieran abordado la pregunta de investigación. Este procedimiento transitó necesariamente por los criterios de inclusión y exclusión de los trabajos.

Partiendo de los criterios definidos previamente, se procedió a realizar una búsqueda bibliográfica lo más amplia posible con el fin de identificar los estudios que pueden cumplir con los criterios de selección. Esta búsqueda, tal como fue planteado en la estrategia inicial, se realizó mediante SCOPUS a fin de garantizar la formalidad de los trabajos y se dividió en tres etapas.

- En la primera etapa se aplicó la estrategia de búsqueda para identificar los posibles trabajos primarios y excluir los duplicados.
- En la segunda etapa se seleccionaron los trabajos que en base a su título y resumen aplicaban según los criterios de inclusión y exclusión previamente definidos.
- En la tercera etapa cada potencial trabajo primario seleccionado se examinó a texto completo para decidir sobre su inclusión o exclusión.

Una vez seleccionados los trabajos que cumplieran con los criterios establecidos y resultaran consonantes con las preguntas de investigación, fueron examinados para determinar las características metodológicas de cada uno como, por ejemplo, el diseño del mismo, verificar si se trata de un trabajo experimental o un metaanálisis de la temática. Dicha averiguación permitió una clasificación de los mismos estableciendo una relación con los resultados de la investigación,

Esta información se documentó por medio de la creación de una planilla en la que los trabajos primarios son las filas y las columnas son las variables que moderan y clasifican los

resultados de la selección de estudios, permitiendo una descripción de sus características y la codificación de los trabajos.

5.3 Procesamiento y extracción de los datos

El objetivo de esta etapa es registrar con precisión la información que se obtiene de los estudios primarios. Se realiza la ejecución de lo planteado en nuestra planificación y en este punto se cuenta con los trabajos potencialmente relevantes para ser evaluados.

Los estudios que investigan o bien plantean la temática referida a la transición de lenguajes de programación visuales a lenguajes basados en texto fueron incluidos.

Se han excluido los estudios que no describen técnicas de transición y que no se considera su aplicación en el contexto de la revisión.

Se realizó la actividad de selección de estudio en tres fases, descritas a continuación:

Inicialmente, se aplicó la estrategia de búsqueda para identificar posibles estudios primarios y se excluyeron los artículos duplicados. Sobre el total de la información, se realizó un refinamiento de la búsqueda anterior con el fin de referenciar toda aquella información relacionada al uso de bloques, tanto desde un componente de la robótica educativa como también en la programación visual, y para ello se aplicaron 3 etapas de selección.

5.3.1 Primera etapa

Sobre el total del contenido, se buscó encontrar toda información que poseyera la palabra “robotic”.

Tras dicha acción se obtuvieron 457 coincidencias, las cuales quedaron representadas en 244 trabajos que cumplían la consigna.

5.3.2 Segunda Etapa

Sobre el total del contenido, se buscó encontrar todo documento que hiciera referencia a la palabra “visual languages”.

Las ocurrencias encontradas con dicha acción fueron 425, que se trasladan a un total de 269 documentos.

La intersección entre los filtros de la etapa 1 y 2, dieron como resultado 27 trabajos que contienen ambas características.

El total de trabajos para esta etapa tiene como resultante, 513 trabajos incluidos y 5098 trabajos excluidos.

En la etapa 2 también se realizó una proyección aplicando los criterios de inclusión y exclusión sobre títulos y abstracts para decidir si incluir o excluir un estudio.

El resultado de la etapa 2 presenta 71 trabajos incluidos y 442 trabajos excluidos.

5.3.3 Tercera etapa

En la etapa 3, se examinó a texto completo cada estudio primario incluido en la selección preliminar de 71 trabajos. Estos se revisaron en detalle para decidir sobre su inclusión o exclusión.

Los estudios primarios incluidos en la selección final se muestran en la Tabla 11 y corresponden a los trabajos relevantes que cumplen con la pregunta de investigación.

El resultado de esta etapa fue de 33 trabajos incluidos y 38 trabajos excluidos. A continuación, en la figura 1 mostramos la distribución de los trabajos incluidos y excluidos.

Figura 1

Trabajos incluidos y excluidos

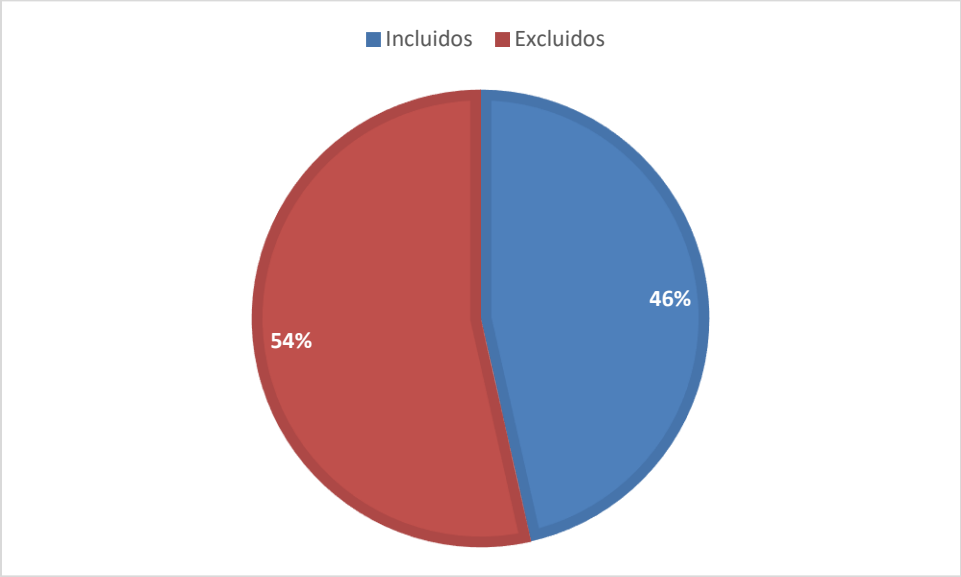


Tabla 11

Trabajos incluidos en la revisión sistemática.

Nro.	Título	Autor	Año	País
1	Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study	Zhen Xua et al.	2019	Estados Unidos
2	The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages	Chen et al.	2018	Estados Unidos

3	Transitioning from Block-based to Text-based Programming Languages	Paul Denny et al.	2018	Nueva Zelanda
4	On Difficulties with Knowledge Transfer from Visual to Textual Programming	Tomas Tóth et al.	2018	Eslovaquia
5	Blue, bluej, greenfoot: Designing educational programming environments	Kölling	2016	Reino Unido
6	Gamesonomy vs scratch: Two different ways to introduce programming	Santamaría et al.	2018	España
7	Developing Educational 3D Games With StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	Boldbaatar et al.	2019	Turquia
8	All rosy in scratch lessons: No bugs but guts with visual programming	Pia Niemela	2017	Finlandia
9	How do Scratch Programmers Name Variables and Procedures?	Alaaeddin Swidan	2017	Holanda
10	Mediated transfer from visual to high-level programming language	Krpan et al.	2017	Croacia
11	Easing the blocks-Text transition	Robinson	2016	Estados Unidos
12	Evaluation of a frame-based programming editor	Price et al.	2016	Australia
13	Initialization in scratch: Seeking knowledge transfer	Dwyer et al.	2016	Estados Unidos
14	Lessons learned from teaching scratch as an introduction to object-oriented programming in delphi	Sukie van Zyl et al.	2016	Sudáfrica
15	From scratch to "Real" programming	Armoni et al.	2015	Israel
16	Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs	Weintrop et al.	2015	Estados Unidos
17	Scratch: A way to logo and python	Dorling et al.	2015	Estados Unidos
18	Language migration in non-CS introductory programming through mutual language translation environment	Matsuzawa et al.	2015	Japon

19	Effect of a 2-week scratch intervention in CS1 on learners with varying prior knowledge	Mishra et al.	2014	Suecia
20	Filling the gap in programming instruction: A text-enhanced graphical programming environment for junior high students	Grace Ngai	2009	China
21	What about a simple language? Analyzing the difficulties in learning to program	Mannila et al.	2014	Finlandia
22	Using Jeroo to introduce object-oriented programming	Sanders et al.	2003	Estados Unidos
23	Using a visual programming environment and custom robots to learn C programming and K-12 STEM concepts	Krishnamoorthy et al.	2016	Estados Unidos
24	TRIK studio: Technical introduction	Mordinov et al.	2017	Rusia
25	Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level	Mladenović et al.	2018	Croacia
26	blockPy: An Open Access Data-Science Environment for Introductory Programmers	Bart et al.	2017	Estados Unidos
27	Learning to communicate computationally with Flip: A bi-modal programming language for game creation	Howland et al.	2014	Reino Unido
28	UNC++Duino: A kit for learning to program robots in python and C++ starting from blocks	Benotti et al.	2017	Argentina
29	Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms	Weintrop et al.	2019	Estados Unidos
30	Frame-based editing: Combining the best of blocks and text programming	Brown et al.	2016	Reino Unido
31	Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming	Kölling et al.	2015	Reino Unido
32	From App Inventor to Java: A Strategy for Mediating the Transition	Tóth et al.	2018	Eslovaquia
33	Pytch — an environment for bridging block and text programming styles	Strong et al.	2021	Irlanda

5.4 Evaluación de calidad

La presente evaluación es relevante y se realizó durante el procedimiento de extracción de datos para garantizar la calidad de los trabajos encontrados. Se definió una medida de ponderación que nos brindara garantías al momento de síntesis de los datos.

Con el objetivo de analizar la calidad de los estudios primarios incluidos se desarrolló para cada pregunta de la lista de verificación una escala de satisfacción en su respuesta.

- Si (S) - 1 punto: Satisface la pregunta de evaluación.
- No (N) - 0 puntos: No satisface la pregunta de evaluación.
- Parcialmente (P) - 0,5 puntos: satisface parcialmente la pregunta de evaluación.
- Experimento (Exp)

Asimismo, también se utilizan una serie de puntuaciones para determinar el grado de satisfacción del trabajo en general.

El total de puntos que obtuvo cada estudio primario tras la ejecución de la lista de preguntas verificación determinó la calidad de este según la escala que se muestra a continuación:

Tabla 12

Resultados evaluación de calidad

Trabajo	Exp.	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4	Score
1	Exp	S	P	P	N	2
2	N	P	P	S	P	2.5

3	Exp	P	P	S	P	2.5
4	N	S	P	S	N	2.5
5	Exp	S	P	S	P	3
6	N	S	S	P	S	3.5
7	N	S	P	S	N	2.5
8	Exp	S	S	N	S	3
9	Exp	S	S	S	P	3.5
10	Exp	S	S	S	N	3
11	N	S	P	P	N	2
12	Exp	S	S	S	S	4
13	Exp	S	S	S	S	4
14	Exp	S	S	S	S	4
15	Exp	S	S	S	P	3.5
16	Exp	S	S	P	N	2.5
17	Exp	S	P	S	P	3
18	N	S	S	S	S	4
19	N	S	S	N	S	3
20	Exp	S	S	N	S	3
21	Exp	S	S	P	S	3.5
22	Exp	S	S	P	S	3.5
23	N	P	P	S	P	2.5
24	Exp	S	S	P	S	3.5
25	Exp	P	S	N	S	2.5
26	N	P	S	P	S	3

27	Exp	S	P	S	P	3
28	Exp	S	S	N	S	3
29	Exp	P	P	P	S	2.5
30	Exp	P	P	P	S	2.5
31	Exp	S	S	S	N	3
32	N	S	S	P	S	3.5
33	N	S	S	N	S	3.5

Capítulo 6 – Resultados y análisis de los datos

En el sexto capítulo se muestran los resultados obtenidos durante la ejecución de nuestra revisión sistemática y las conclusiones a las que hemos arribado mediante el análisis de dichos datos.

5.1 Análisis de los resultados

Se realiza la síntesis e interpretación de los resultados obtenidos luego de la ejecución de la revisión sistemática de la literatura, exhibiendo a través de análisis cuantitativos y cualitativos los hallazgos principales de la evaluación de los estudios primarios.

5.2 Análisis cuantitativo

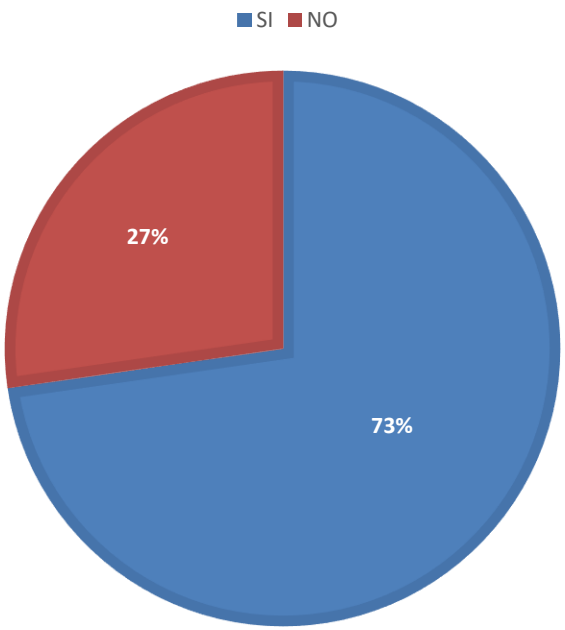
El análisis cuantitativo propone contar los resultados de los trabajos analizados, siguiendo los lineamientos definidos en la pregunta principal de la investigación.

Fueron analizados 33 trabajos relacionados con la transición entre lenguajes visuales y lenguajes textuales dentro del ámbito educativo, con el objetivo de evaluar si estos trabajos proponen una estrategia o herramienta de software (IDEs, lenguajes, motor de juegos, métodos, librerías, etc.) que permitan mitigar la problemática planteada en la presente investigación.

A continuación, presentamos la siguiente distribución, donde podemos observar que un 73% (24 trabajos) presentan herramientas de software que permiten realizar la transición de un lenguaje visual a un lenguaje textual, mientras que un 27% (9 trabajos) no presentan herramientas de software. Este último grupo de trabajos, como veremos con más detalle, presentan métodos educativos, estudios de casos o experimentos.

Figura 2

Trabajos que presentan herramientas de software



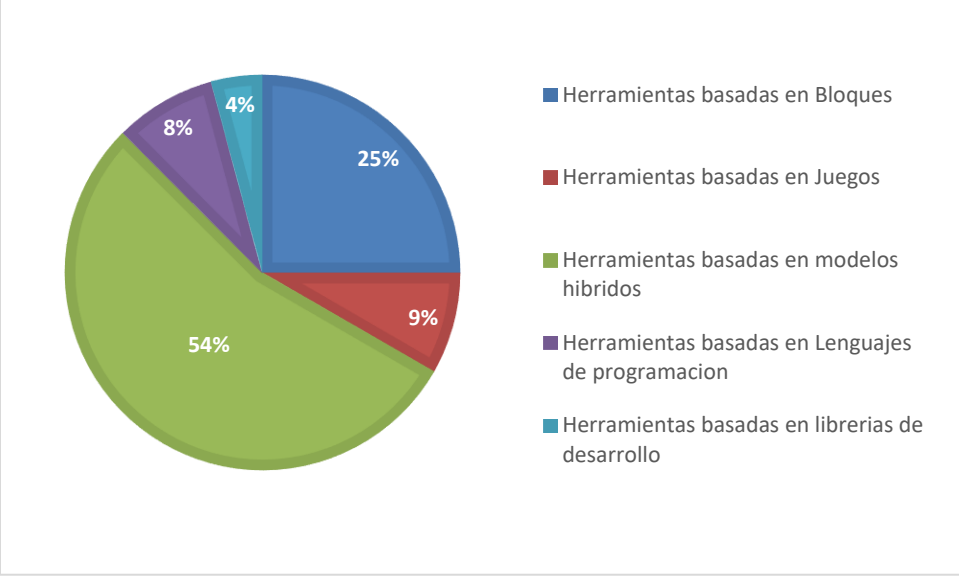
5.2.1 Análisis de los trabajos que presentaron herramientas de software.

A continuación, se analiza el conjunto de trabajos que presentan herramientas de software para abordar la problemática.

En respuesta a la sub pregunta de investigación "¿Qué herramienta de software propone el trabajo?" hallamos 24 trabajos que presentan herramientas de software: 6 trabajos están basados en herramientas de bloques, 2 trabajos en herramientas basadas en juegos, 13 trabajos basados en herramientas híbridas de desarrollo y 1 trabajo se basa en la utilización de una librería de desarrollo. La figura 3 muestra la distribución de los diferentes tipos de herramientas de software encontrados.

Figura 3

Tipos de herramientas de software.

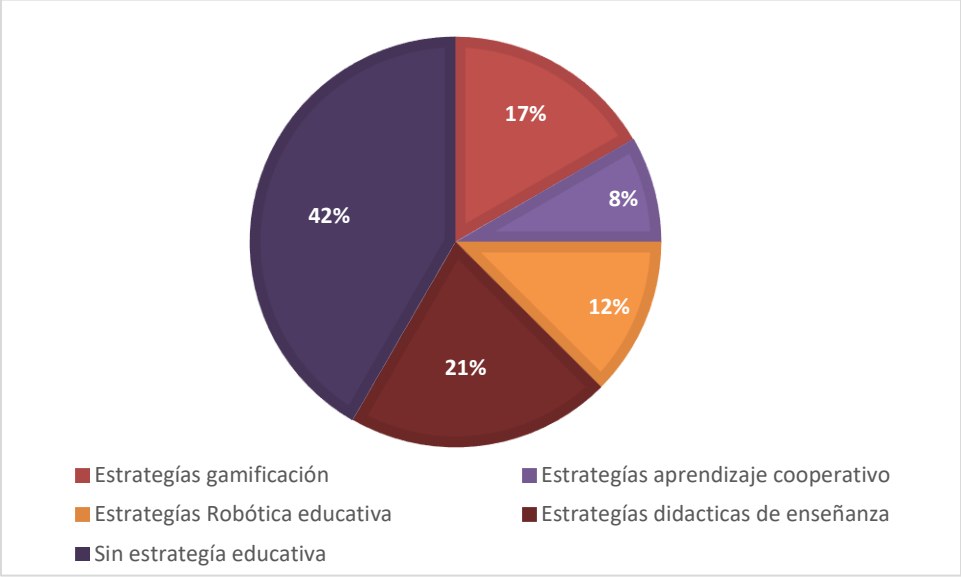


Con respecto a la sub pregunta de investigación "¿Propone el trabajo una estrategia educativa para acompañar la transición de un lenguaje visual a uno textual?" encontramos 14 trabajos que presentan estrategias educativas para acompañar la utilización de la herramienta de software propuesta para abordar la problemática de la transición entre LPV y LPT; de los cuales 4 trabajos incluyen estrategias basadas en técnicas de gamificación, 2 trabajos técnicas de aprendizaje colaborativo, 3 trabajos utilizan la robótica educativa como estrategia educativa para el aprendizaje y 5 trabajos utilizan estrategias educativas basadas en técnicas didácticas de enseñanza. En contrapartida 10 trabajos no presentan estrategias educativas que acompañen la transición entre lenguajes de programación.

La figura 4 muestra la distribución de las estrategias educativas propuestas encontradas.

Figura 4

Estrategias educativas reportadas en los trabajos.



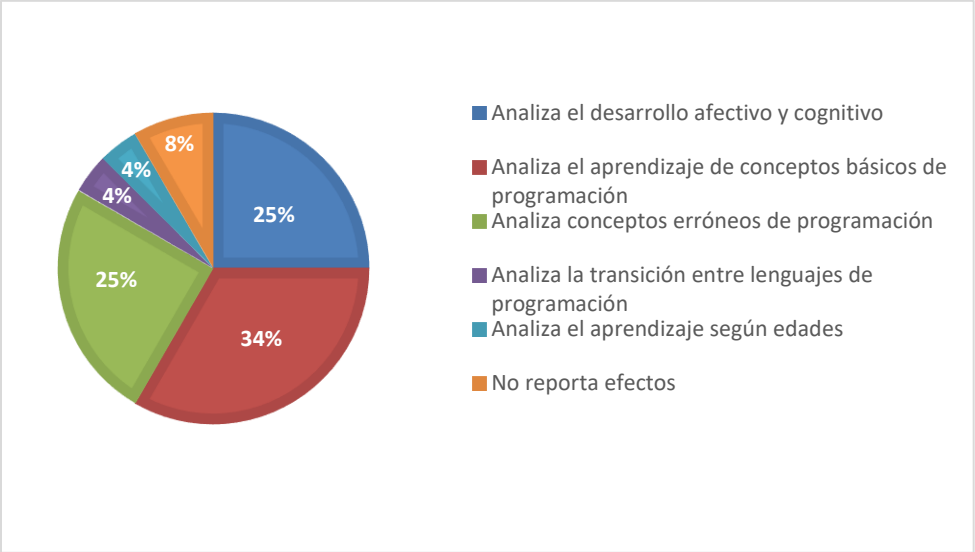
En relación a la sub pregunta de investigación "¿Qué efectos analiza o reporta el estudio?" encontramos 22 trabajos que analizan o reportan efectos tras la utilización de herramientas de

software. Respecto a los trabajos, 6 de ellos analizan el desarrollo afectivo y cognitivo; 8 trabajos se enfocan en el aprendizaje de conceptos básicos de programación; 6 trabajos analizan los efectos de los conceptos erróneos de programación; 1 trabajo analiza los efectos del aprendizaje según edades y 1 trabajo analiza efectos al realizar la transición entre LPV y LPT. Por otra parte, 2 trabajos no analizan ni reportan ningún efecto.

La figura 5 muestra la distribución de los efectos reportados por los estudios analizados.

Figura 5

Efectos reportados en los trabajos.



Respecto a la sub pregunta de investigación "¿Qué nivel del ciclo educativo se involucra en el estudio?", primero indicaremos los diferentes niveles educativos según las edades de los estudiantes:

- Nivel Elemental - 4 a 5 años.
- Nivel Primario - 6 a 10 años.
- Nivel Secundario - 11 a 17 años.

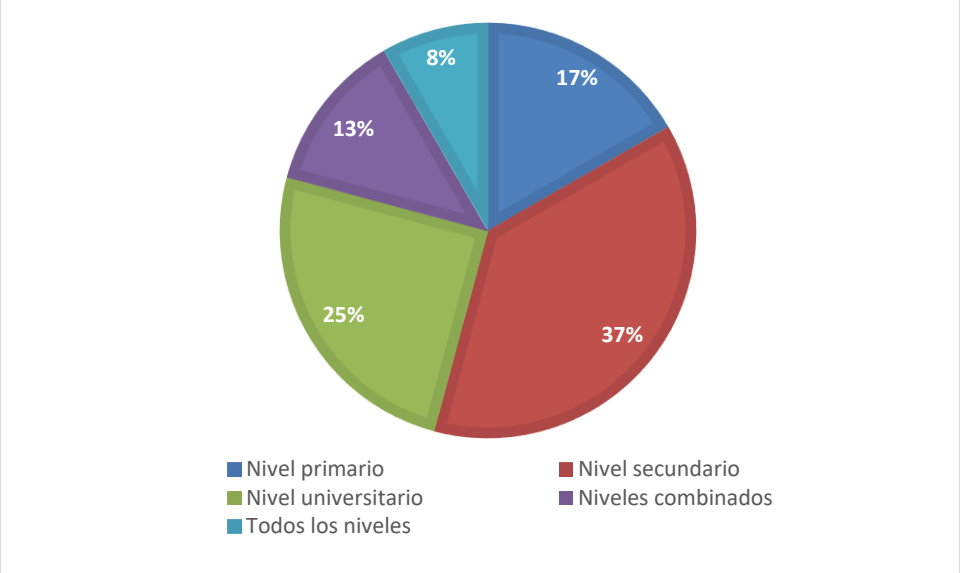
- Nivel Universitario - desde 18 años.

A continuación, indicamos los trabajos que informan el nivel educativo en su estudio y su distribución: 4 trabajos se centran en el nivel primario, 9 trabajos sobre el nivel secundario y 6 trabajos en el nivel universitario. Por otra parte, encontramos 3 trabajos que combinan niveles, a razón de nivel elemental y nivel primario o nivel primario y nivel secundario. A su vez, encontramos 2 trabajos cuyo estudio abarca todos los niveles educativos.

La figura 6 muestra la distribución de los niveles educativos informados en los trabajos.

Figura 6

Niveles educativos informados en los trabajos.



5.2.2 Análisis de los trabajos que no presentaron herramientas de software

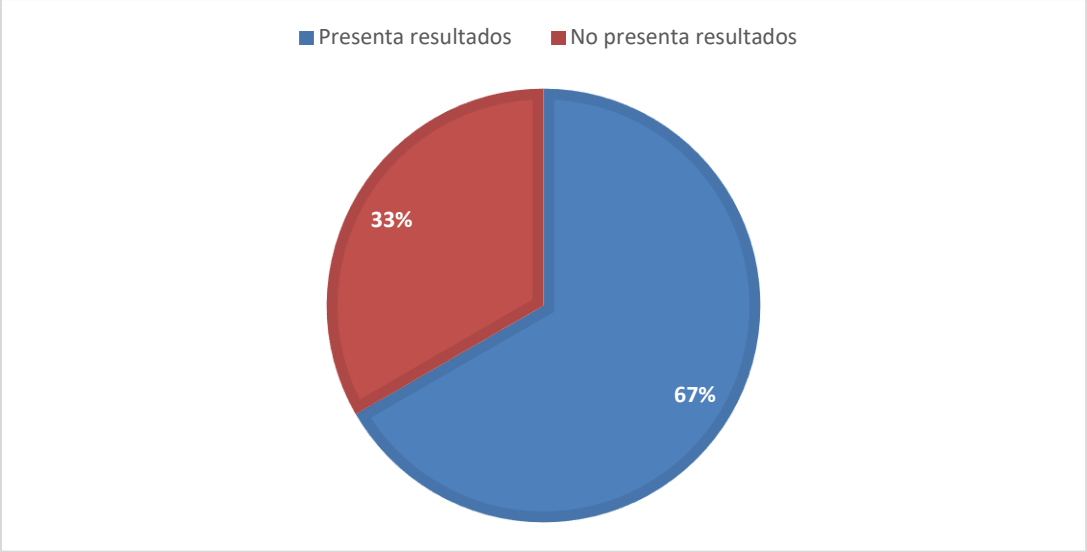
A continuación, se analiza el grupo de trabajos que no presentan herramientas de software para abordar la problemática. Dicho grupo suma 9 trabajos.

En respuesta a la sub pregunta de investigación “¿El trabajo presenta resultados de algún experimento donde se ensaye la transición de un lenguaje visual a uno basado en texto?” hallamos 6 trabajos que presentan resultados de experimentos o análisis de casos, mientras que 3 no reportan resultados en su estudio.

La figura 7 muestra la distribución de los estudios.

Figura 7

Trabajos que presentan resultados de prueba.



En relación con la sub pregunta de investigación “¿Propone el trabajo una estrategia educativa para acompañar la transición de un lenguaje visual a uno textual?” encontramos 3 trabajos que presentan estrategias educativas para acompañar la transición entre lenguajes de programación, mientras que 6 trabajos no presentan estrategias educativas

La figura 8 muestra la distribución de las estrategias educativas propuestas encontradas.

Figura 8

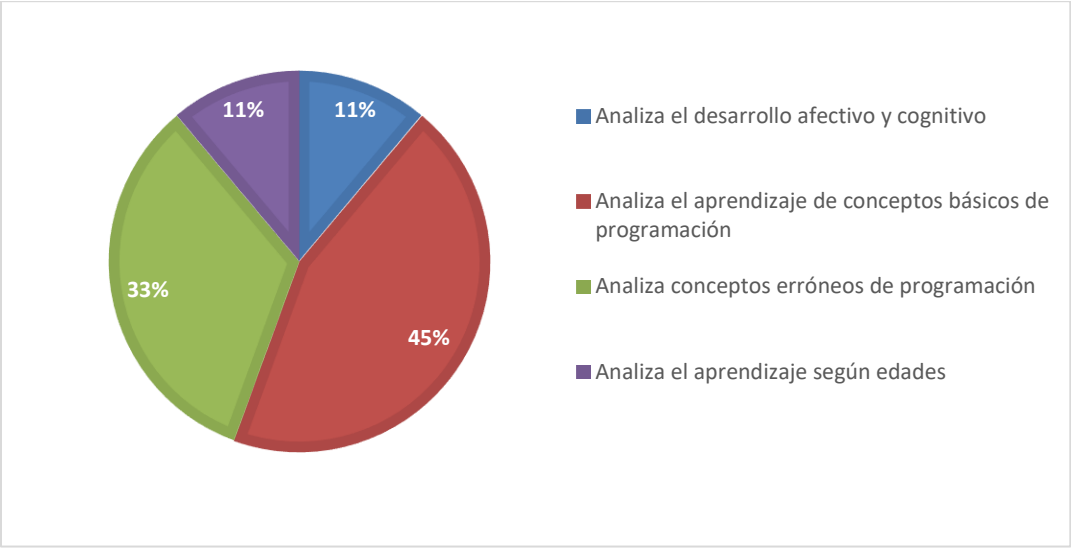
Trabajos que presenta estrategias educativas.

Respecto a la sub pregunta de investigación "¿Qué efectos analiza o reporta el estudio?" encontramos 1 trabajo que analiza el desarrollo afectivo y cognitivo; 4 trabajos se enfocan en el aprendizaje de conceptos básicos de programación; 3 trabajos analizan los efectos de los conceptos erróneos de programación y, por último, 1 trabajo analiza los efectos del aprendizaje según edades.

La figura 9 muestra la distribución de los efectos reportados por los estudios analizados.

Figura 9

Efectos reportados por los estudios analizados.



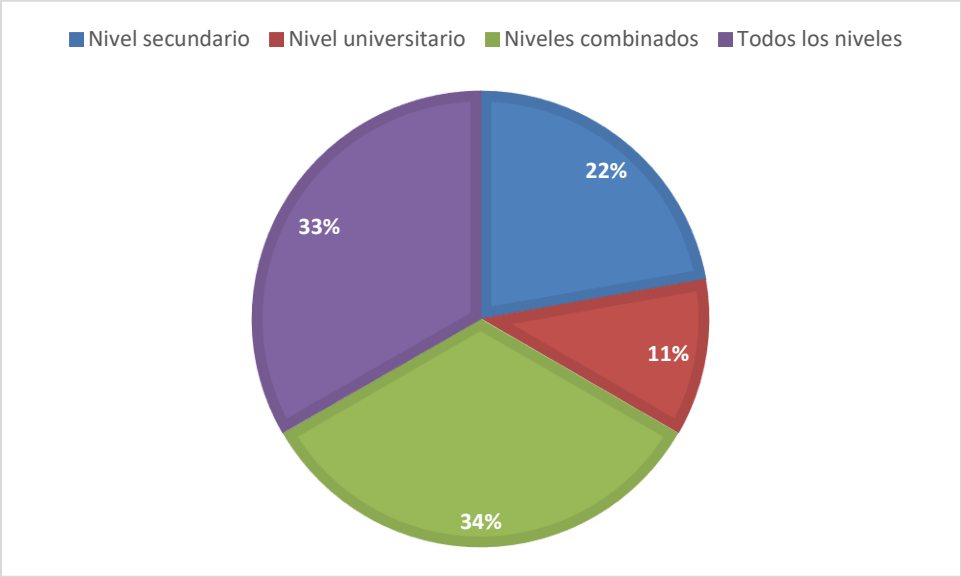
En cuanto a la sub pregunta de investigación "¿Qué nivel del ciclo educativo se involucra en el estudio?" se indican los diferentes niveles educativos según las edades de los estudiantes:

- Nivel elemental - 4 a 5 años.
- Nivel primario - 6 a 10 años.
- Nivel Secundario - 11 a 17 años.
- Nivel Universitario - desde 18 años.

A continuación, se presentan los trabajos que informan el nivel educativo en su estudio y su distribución. Se encontraron 2 trabajos sobre el nivel secundario y 1 trabajo en el nivel universitario. Por otra parte, encontramos 3 trabajos que combinan niveles, a razón de nivel elemental y nivel primario o nivel primario y nivel secundario. Finalmente, hallamos 3 trabajos cuyo estudio abarca todos los niveles educativos.

Figura 10

Nivel educativo informado en el trabajo.



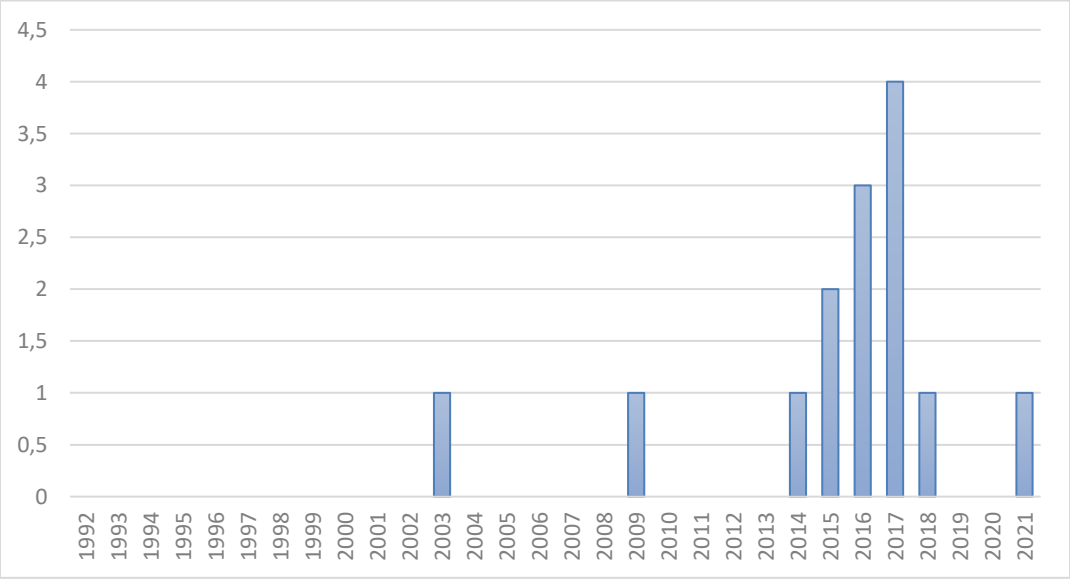
5.3 Análisis cualitativo

En este apartado apoyándonos en los resultados cuantitativos obtenidos se evaluarán los trabajos cualitativamente, focalizando el análisis en aquellos que presentaron una mayor coincidencia respecto a nuestra pregunta de investigación principal "¿Cuál es la estrategia educativa más utilizada para realizar la transición entre lenguajes de programación visuales y textuales surgida en los últimos 30 años?"

Tal como se puede verificar en la figura 3, el 54% de los trabajos analizados plantean la utilización de un enfoque híbrido de desarrollo. Este enfoque combina elementos visuales y textuales de los lenguajes. Luego, al analizar cada trabajo por su año de publicación, nos demuestra que el mayor impacto se obtuvo durante la última década, siendo la estrategia de mayor popularidad para abordar la problemática de transición entre lenguajes de programación. En la figura 11 se observa la distribución de los estudios.

Figura 11

Distribución de los trabajos según el año de publicación.



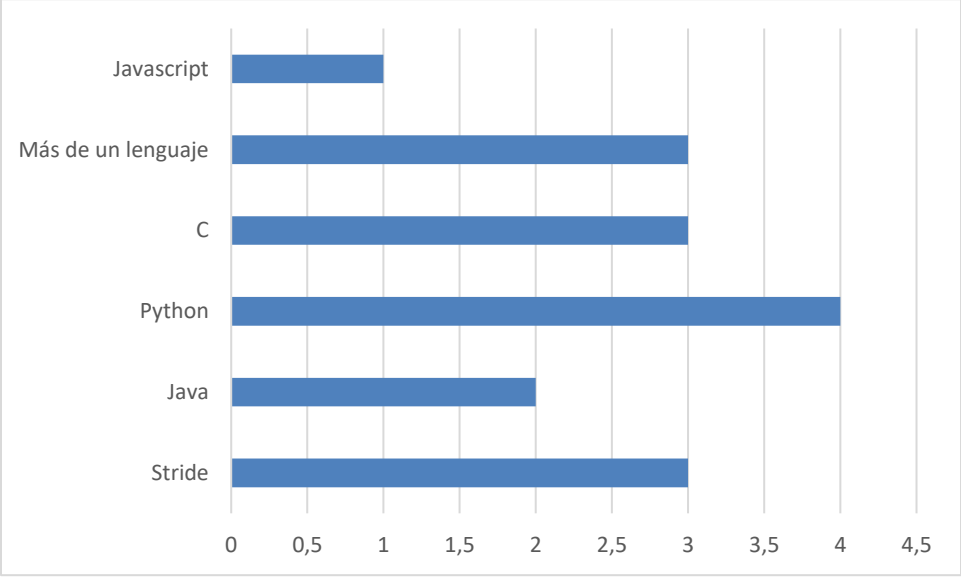
Otro aspecto a considerar son las herramientas de software y los lenguajes de programación que acompañan a la modalidad híbrida de desarrollo para permitir a los programadores transitar entre lenguajes visuales y textuales.

Según el relevamiento de los estudios, pudimos observar que el lenguaje de programación textual más elegido al momento de realizar la transición es Python, dado que se trata de un lenguaje de alto nivel del cual se considera posee una sintaxis simple en comparación con otros lenguajes de programación textuales.

A continuación, en la figura 12 observamos la distribución de lenguajes de programación textuales.

Figura 12

Distribución de los lenguajes de programación informados en los estudios.



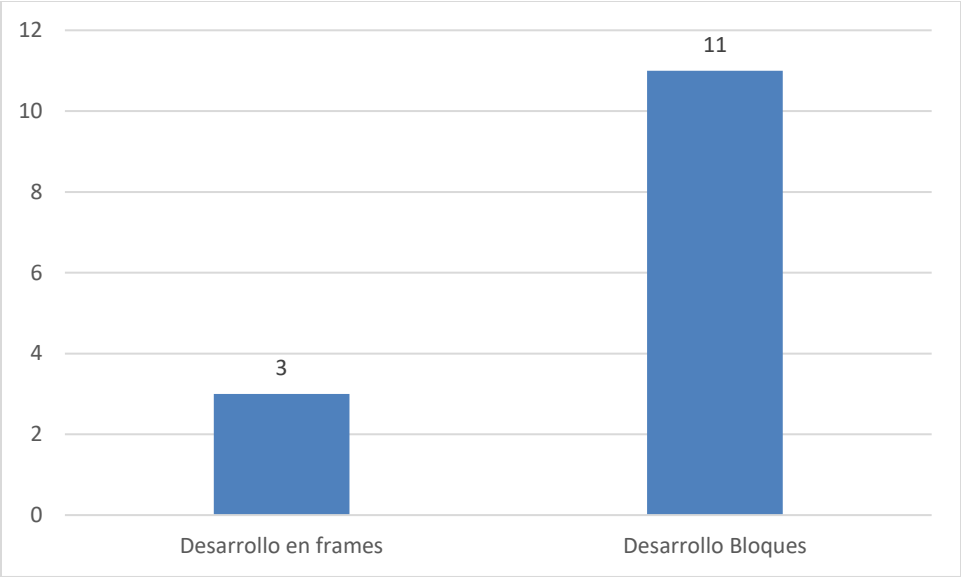
Un segundo aspecto a considerar es el lenguaje de programación visual utilizado en el entorno de desarrollo híbrido. En relación con este punto, observamos que la opción más

predominante es la utilización de bloques de desarrollo, aunque en menor proporción una alternativa incipiente es el desarrollo basado en frames.

En la figura 13 observamos su distribución en los estudios.

Figura 13

Distribución de lenguajes visuales.



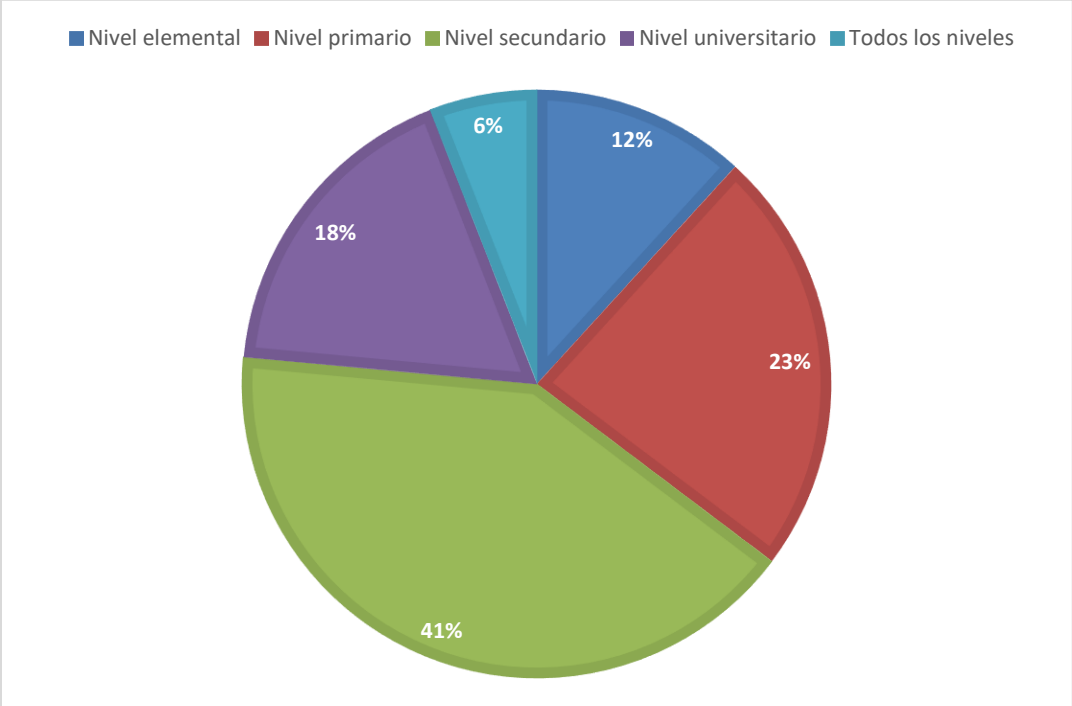
Por último, analizamos el nivel de escolaridad de los programadores y su relación con el uso de entornos híbridos de desarrollo, dado que consideramos que se vuelve fundamental para la aplicación de esta estrategia determinar cuál debería ser la edad de inducción de los programadores.

En la figura 14 se muestra la distribución de los niveles educativos donde se utilizan estrategias de desarrollo híbrido. Siendo el nivel secundario el más preponderante, comprendiendo edades que oscilan entre los 11 y los 17 años, el resultado cobra sentido por tratarse de niveles superiores de educación donde la utilización de lenguajes de programación textuales es habitual, debido a que es parte de la preparación profesional de los programadores.

A partir de la presente muestra podemos interpretar que durante el nivel secundario los estudiantes ostentan mayores posibilidades de dar el salto desde la programación visual hacia la programación textual mediante la utilización de estrategias basadas en entornos híbridos de desarrollo.

Figura 14

Utilización de entornos híbridos de desarrollo según nivel educativo.



Capítulo 7 - Conclusión y trabajos futuros.

En el séptimo capítulo se exponen la conclusión y las líneas futuras de investigación.

7.1 Conclusión

La investigación que hemos realizado bajo el formato metodológico de una revisión sistemática de la literatura nos brindó información empírica sobre la transición entre lenguajes de programación visuales y textuales.

Tal como hemos visto, el aprendizaje de un lenguaje de programación textual puede llegar a ser frustrante en algunos casos debido a las complejidades de sus conceptos y la sintaxis particular de estos, causando que muchos estudiantes no estén predispuestos a su aprendizaje.

Lo contrario ocurre con los lenguajes de programación visuales, muy utilizados en el contexto educativo para introducir a los estudiantes en programación, ya que, mediante interfaces visuales amigables, se facilita su uso y se permite que los estudiantes encuentren en la programación componentes afectivos positivos tales como la confianza o motivación potenciando su aprendizaje.

Por otra parte, ocurre que los conocimientos, habilidades y experiencias obtenidas tras la utilización de lenguajes visuales no resultan suficientes para que los programadores novatos puedan desenvolverse en un entorno de programación profesional utilizando lenguajes de programación textuales. He aquí que se genera una brecha al realizar la transición entre un lenguaje visual y un lenguaje textual, denotando dificultades para resolver tareas, bien sea por falta de capacidades de abstracción dada la naturaleza de los conceptos de programación o por la adquisición de conceptos erróneos.

Por este motivo, surgió en las últimas décadas la necesidad de apoyar a los estudiantes durante la transición de la programación visual hacia la textual o viceversa. Para ello, como pudimos relevar durante nuestra investigación, se han probado distintos enfoques que implican estrategias pedagógicas, modificaciones de herramientas de software o la combinación de ambas. También comprobamos que surgieron nuevos enfoques de desarrollo que ocupan las características de ambos lenguajes y que, a su vez, cubren las carencias de estos al momento del aprendizaje, denominados *entornos híbridos de desarrollo*.

Los entornos híbridos de desarrollo son parte de una estrategia educativa incipiente, pero que, al mismo tiempo, poseen una gran adopción al momento de seleccionar una herramienta para llevar a cabo la transición entre lenguajes visuales y textuales.

En esta investigación se pudo comprobar la hipótesis planteada, esto fue confirmado en el presente trabajo al verificar que desde 1990 a la actualidad el 54% de los estudios se basan en su utilización y que, a su vez, se encuentran en constante evolución para consolidarse como un nuevo paradigma de programación.

Consideramos que el uso de este tipo de entornos es beneficioso ya que utilizan programación basada en componentes visuales que liberan a los estudiantes de problemas sintácticos y, al mismo tiempo, los exponen a la sintaxis de programación, asimilándola a través del aprendizaje directo como consecuencia de su uso.

Al combinar las mejores características de ambos enfoques de programación (visual y textual) junto con la incorporación de distintas técnicas didácticas de aprendizaje, o bien integrándose con la robótica educativa, se fortalece la transición y se minimiza la problemática. En nuestra investigación observamos que las estrategias que utilizan técnicas didácticas de enseñanza representan el 21% de la muestra, mientras que el uso de la robótica educativa para

acompañar la transición representa el 12%. Estos porcentajes son relevantes y cobran notoriedad dado que el 42% de la muestra no presenta estrategias educativas que acompañen la utilización del modelo híbrido de desarrollo.

Finalmente, consideramos que la edad de los estudiantes al momento de realizar la transición entre lenguajes de programación es importante, dado que, en los niveles de educación elemental y primario, los estudiantes poseen entre 4 y 10 años y prepondera el uso de lenguajes visuales de programación. En los niveles secundario (11 a 17 años) y universitario (17 años) en adelante se utilizan lenguajes de programación textuales. En la presente investigación verificamos que el 37% de los estudios reflejan el uso de estrategias híbridas de desarrollo en el nivel secundario, coincidimos con dichos resultados y consideramos que es apropiado profundizar en dicho nivel, dado que los niveles de educación superior manejan lenguajes profesionales de desarrollo basados en texto, siendo el nivel secundario un espacio adecuado para llevar a cabo la utilización de entornos híbridos de desarrollo para abordar la transición entre lenguajes de programación visuales y textuales.

7.2 Líneas futuras de investigación

Las líneas futuras de investigación que encontramos al finalizar la revisión sistemática de la literatura para mejorar y desarrollar estrategias que ayuden a solventar el problema de la transición entre lenguajes de programación textuales y visuales deben orientarse a adaptar las técnicas didácticas de enseñanza al plan de estudios de programación.

En segundo lugar, se debe continuar investigando sobre el enfoque de programación basado en frames, profundizando en la relación entre estos tres tipos de editores: bloques, frames

y texto, dado que se trata de una alternativa prometedora como paso de transición entre bloques y texto.

7.3 Anexos

Base de datos exportada de SCOPUS:

https://docs.google.com/spreadsheets/d/1qycgut4Yfm13QAO3suEkCNeEUbrhe8Vb0wTxzMRKpXY/edit?usp=sharing.

7.4 Acrónimos

LPV Lenguaje de Programación Visual.

LPT Lenguaje de Programación Textual.

IDE Entorno Integrado de Desarrollo.

CT Pensamiento Computacional.

EVP Entornos Visuales de Programación.

GPL Lenguajes de propósito general.

DSL Lenguajes de dominio específico.

Referencias

Algaraibeh, Dousay and Jeffery (2020), *Integrated Learning Development Environment for Learning and Teaching C/C++ Language to Novice Programmers*, IEEE Frontiers in Education Conference (FIE), 2020, pp. 1-5, doi: 10.1109/FIE44824.2020.9273887.

Babkin (2012), *Visual programming languages*, University of Tartu, Programming Languages Research Seminar (MTAT.03.271)

Cardenas Varon (2009), *Un lenguaje para composición de conjuntos*, Universidad de los Andes. https://repositorio.uniandes.edu.co/handle/1992/10968

Cheung. Ngai, Grace, Chan, Winnie. (2009). Filling the gap in programming instruction. ACM SIGCSE Bulletin. 41. 276. 10.1145/1508865.1508968.

Cilliers, Calitz. y Greyling. (2005). *El efecto de integrar una notación de programación icónica en cs1*. Boletín ACM SIGCSE, ACM, 37, 108–112.

Cooper (2010). *El diseño de alice. ACM Transactions on Computing Education* (TOCE), 10(4)

Denny, AL-R. (2011). *Comprender la barrera de sintaxis para principiantes*. ITICSE. Darmstadt.

Federico (2011). *Una implementación mínima, extensible y de arrastrar y soltar del lenguaje de programación c.* Actas de la conferencia de 2011 sobre educación en tecnología de la información, págs. 191–196. ACM Garlick, R., Cankaya

Gabbrielli, Martini (2010*), How to Describe a Programming Language. In: Programming Languages: Principles and Paradigms*. Undergraduate Topics in Computer Science. Springer, London. https://doi.org/10.1007/978-1-84882-914-5_2

Ghezzi (1996), Programming language concepts , Politecnico di Milano Mehdi Jazayeri, Technische Universität Wien

Katz (2016), *Entornos digitales y políticas educativas: dilemas y certezas* (pp.17-58). IIPE-UNESCO.

Kitchenham, et al. (2009), "Systematic Literature Reviews in Software Engineering – A Systematic Literature Review.", Information and Software Technology 51.1 7-15.

Knaggs, Welsh (2019), ARM Assembly Language Programming, School of design, engineering and computing, Bournemouth University, UK.

Koitz, Slany. (2014). *Comparación empírica de la manipulación de fórmulas visuales e híbridas en lenguajes de programación educativos para adolescentes*. Actas del 5º Taller sobre Evaluación y Usabilidad de Lenguajes y Herramientas de Programación (págs. 21–30).

Kölling (2010). The Greenfoot Programming Environment. ACM Transactions on Computing Education , 10 (4).

Kuhail et al. (2021), *Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review*. Zayed University, United Arab Emirates., DOI 10.1109/access.2021.3051043

Matsuzawa, Ohata, Sugiura y Sakai. (2015). *Migración de idiomas en programación introductoria que no es CS a través de un entorno de traducción de idiomas mutuo*. En: Actas del 46º Simposio Técnico de ACM sobre Educación en Ciencias de la Computación (págs. 185–190).

Parveen and Fatima (2016), Performance Comparison of Most Common High Level Programming Language, International Journal of Computing Academic Research, 5(5), 246-258.

Red Hat (9 de enero de 2019). *El concepto de IDE*. https://www.redhat.com/es/topics/middleware/what-is-ide

Rodríguez, Parra, Dolz, y Ramírez (14-18 de octubre de 2019). *Transición desde programación basada en bloques a basada en texto: una revisión del campo.* XXV Congreso Argentino de Ciencias de la Computación, Argentina.

Sandoval-Reyes, Galicia-Galicia y Gutierrez-Sanchez (2011*). Entornos de aprendizaje visual para la programación informática*. Conferencia de Electrónica, Robótica y Mecánica Automotriz (CERMA) pp 439–444).

Satav et al. (2011), A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development, Conference on IT - Contemporary & Future Technologies for Social Change, India.

Selby y Woollard (2013) *Pensamiento computacional: la definición en desarrollo*. http:// eprints.soton.ac.uk/356481

Slany (2012). Catroid: *Un sistema de programación visual móvil para niños. En: Actas de la 11ª Conferencia Internacional sobre Diseño de Interacción y Niños* (pp. 300–303).

Tejera-Martínez, Aguilera y Vílchez-González (2020). *Lenguajes de programación y desarrollo de competencias clave.* Revisión sistemática. Revista Electrónica de Investigación Educativa, 22, e27, 1-16. https://doi.org/10.24320/redie.2020.22.e27.2869

Van Zyl, Mentz y Havenga (2016), Lessons Learned from Teaching Scratch as an Introduction to Object-oriented Programming in Delphi, African Journal of Research in Mathematics, Science and Technology Education, DOI: 10.1080/18117295.2016.1189215

Weintrop and Wilensky (2018) How block-based, text-based, and hybrid block/textmodalities shape novice programming practices. International Journal of Child Computer Interaction, 17:83–92

Zhang , Nouri (2019), *A systematic review of learning computational thinking through Scratch in K-9*, Department of Computer and Systems Sciences, Stockholm University, Borgarfjordsgatan 12, Kista, Sweden.