D. Alulema and M. Paredes-Velasco, "Relation Between Programming Visual Learning With VILEP and Students? Emotions" in IEEE-RITA, vol. 16, no. 1, pp. 36-44, FEB. 2021. doi: 10.1109/RITA.2021.3052481

Relación entre el Aprendizaje de la Programación Visual con VILEP y las Emociones de los Estudiantes

Darwin Alulema, y Maximiliano Paredes

***Title*— Relationship between Learning Visual Programming with VILEP and the Emotions of Students**

***Abstract*— In this article a visual programming tool is shown, which allows coding programs by abstracting complex parts of the syntactic structures of the language by means of graphic representations combined with textual expressions. The tool allows student to relate the syntax of a language to the logic of solving a problem. To validate the proposal, we carried out an experience with a group of students who used the tool and another group who used Eclipse. The results show that there was an additional 23.4% of students who used the tool and reduced their error rate compared to those who used Eclipse.**

***Index Terms*— Computer science education, Learning, Emotion recognition.**

I. Introducción

Este trabajo es la extensión del artículo publicado en las actas del Congreso Internacional sobre Aprendizaje, Innovación y Cooperación CINAIC 2019 titulado “Evaluación de las Emociones de los Estudiantes en el Aprendizaje Visual de la Programación” (DOI: 10.26754/CINAIC.2019.0051). El artículo se ha extendido en el presente trabajo de la siguiente forma: a) se ha incorporado una nueva sección de trabajos relacionados (sección II), b) se ha ampliado la descripción del diseño de la herramienta (sección III) y c) se ha ampliado el análisis de los resultados obtenidos (subsección E de la sección V).

El rápido avance de la tecnología ha cambiado la forma en que los profesores enseñan y los estudiantes aprenden. Pese a que estos avances también están presentes en el aprendizaje de la programación [1], el aprendizaje de ésta no es una tarea fácil. Aprender a programar conlleva alcanzar una serie de difíciles logros, tanto para los estudiantes de los grados de informática, como para los estudiantes de otros grados afines. Sin embargo, los primeros, debido al propio ámbito de estudio, deberán desarrollar más esta capacidad [2]. Los trabajos de Aktunc (2013) enumeran numerosos desafíos a los que se enfrentan los instructores y que complican el aprendizaje de la programación en las asignaturas introductorias de los grados de informática y afines: a) gran variación del perfil de conocimiento de los estudiantes; b) desánimo y desmotivación de la mayoría de los estudiantes ya que perciben la programación como una tarea cognitiva difícil y compleja; c) excesivo tiempo destinado a la enseñanza de la sintaxis del lenguaje de programación. Hay que tener en cuenta que pasar demasiado tiempo aprendiendo la sintaxis del lenguaje sin aplicarlo en un contexto de uso es perjudicial para los estudiantes [3]; y d) la mayoría de los entornos de programación utilizados para la enseñanza son confusos, ya que fueron construidos para el desarrollo profesional de software y no tienen un enfoque didáctico, de tal modo que todos estos factores generan problemas en los cursos introductorios de programación [4].

La forma tradicional de introducir la programación para estudiantes que se inician en la misma es a través de un curso introductorio orientado al lenguaje [5]. Pero este enfoque orientado al lenguaje produce varios problemas: a) los estudiantes tienen dificultades por la complejidad de la sintaxis, b) la sintaxis exige un tiempo extra de aprendizaje y c) el lenguaje en sí mismo no aporta ventajas en el entendimiento de los conceptos de programación que subyacen en sus estructuras, incluso su sintaxis puede dificultar la comprensión de los conceptos. Sin embargo, el uso de un lenguaje de programación es necesario para el aprendizaje y práctica de los conceptos de programación. Es precisamente este el motivo por el que frecuentemente la oferta real de un curso de programación de primer año de grado dedicará una considerable cantidad de tiempo de clases al aprendizaje de la sintaxis del lenguaje [6]. Por tanto, no se debe prescindir del lenguaje en el proceso de aprendizaje y se debería buscar soluciones que gestionen de una manera adecuada su uso como instrumento de aprendizaje en el contexto educativo para mitigar los inconvenientes señalados. Esta investigación se centra precisamente en buscar soluciones con esta orientación. Para motivar a los jóvenes en el primer año, la experiencia de aprendizaje debe ser enriquecedora incorporando actividades prácticas, creativas y "divertidas" [7]. Sin embargo, en las asignaturas de introducción a la

D. Alulema, Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador; e-mail: doalulema@espe.edu.ec).

M. Paredes, Universidad Rey Luan carlos, Móstoles, Madrid, España; e-mail: Maximiliano.paredes@urjc.es).

rogramación la percepción que tienen los estudiantes no es precisamente ésta [8]. Los estudiantes, además de enfrentarse a los retos de aprender a formar soluciones estructuradas para los problemas de programación, deben enfrentarse también a la dificultad de la sintaxis y comandos del lenguaje de programación que usan para plasmar las soluciones de los problemas, cuyos comandos pueden tener aparentemente nombres confusos. En este contexto de dificultad para el estudiante, éste suele percibir que no se genera un contexto de aprendizaje personalmente significativo, experimentando una desmotivación y pudiendo incluso desalentarse ante el aprendizaje [5].

El objetivo de esta investigación es proponer recursos educativos para el aprendizaje de la programación, para lo cual los autores proponen abstraer progresivamente la sintaxis del lenguaje de programación (mediante técnica de scaffolding). De esta forma se pretende motivar al estudiante desde un estado emocional positivo. De esta forma, los estudiantes adquirirán un cierto nivel de fluidez en un lenguaje de programación antes de comenzar a implementar sus soluciones en código fuente directamente. Algunas investigaciones proponen que, en lugar de implementar sistemas orientados al aprendizaje completo de un lenguaje de programación, se implementen herramientas u objetos de aprendizaje a pequeña escala para conceptos específicos de programación [6]. Estas herramientas deben integrar técnicas de scaffolding [9] que adapten la estructura y contenidos del aprendizaje al estudiante. La investigación en educación y psicología cognitiva sugieren que muchos estudiantes hoy en día presentan un perfil de aprendizaje visual e interactivo. Por tanto, parece razonable reducir el detalle textual de la sintaxis del lenguaje de programación que tienen que aprender los estudiantes en las primeras etapas, y desarrollar más contenidos visuales sobre los conceptos de programación en sí, intentando facilitar la relación de estos conceptos con el proceso de resolución de problemas.

Por estas razones, en este artículo se propone una herramienta denominada VILEP (VIsual LEarning Object-oriented Programming) que está desarrollada sobre esta idea para el aprendizaje de POO (Programación Orientada a Objetos), la cual puede ocultar o hacer visibles detalles de la sintaxis del lenguaje que utiliza el estudiante combinando representaciones visuales y textuales. De esta forma, combina programación visual con programación textual de código fuente de manera adecuada, orientando así el proceso a un aprendizaje visual. Estudios previos demuestran que el uso del aprendizaje visual tiene un impacto significativo para mejorar la resolución de problemas y habilidades de pensamiento analítico de los estudiantes y promueve el aprendizaje activo [10]. En este sentido existen herramientas de programación con enfoque más o menos visual para introducir a los estudiantes en la programación, como Scratch [11], Alice [12], Blockly [13], Greenfoot [14], entre otras. Sin embargo, las herramientas visuales con el tiempo hacen que los estudiantes se dispersen y provocan distracción de su atención, por este motivo propuestas como la de Rahman [5] señalan no abandonar por completo el uso de lenguajes textuales. Además, este tipo de herramientas no establecen técnicas scaffolding para el aprendizaje de la programación como en esta propuesta. Por esta razón y sin perder de vista que el objetivo de un curso introductorio de programación no debe ser sólo enseñar un lenguaje de programación, sino que debe enseñar además las diferentes formas de resolución de problemas, lógica de razonamiento, diseño básico de algoritmos y conceptos de programación generales, intentando tener poco o mínimo énfasis en la sintaxis del lenguaje [5]. En este contexto, el uso de herramientas de programación visual en las primeras etapas del aprendizaje podría facilitar el aprendizaje. Es así que la herramienta VILEP descrita en este artículo ha sido desarrollada mediante ingeniería dirigida por modelos y propone un editor gráfico que permite a los estudiantes implementar programas en Java mediante recursos visuales ocultando expresiones sintácticas complejas del lenguaje. En este artículo se describe la herramienta desde un enfoque docente y demuestra de forma preliminar la validez de la herramienta en el contexto educativo. Para ello se ha realizado una experiencia con estudiantes de programación de primer año en el aula y se ha medido el conocimiento, percepciones y las emociones que han experimentado con el uso de la herramienta.

El artículo está estructurado de la siguiente manera: la sección II menciona algunos de los trabajos que abordan el empleo de herramientas especializadas en la enseñanza de la programación. En la sección III se describe cómo se implementó la herramienta VILEP empleando ingeniería de modelos. La sección IV aborda el método de aprendizaje que se ha empleado para abordar el uso de la herramienta propuesta. La sección V describe la experiencia realizada en el aula y la sección VI muestra las conclusiones y los trabajos futuros.

II. TRABAJOS RELACIONADOS

La programación se considera una habilidad fundamental ya que muchos de los conceptos de programación se utilizan en casi todos los cursos básicos en los grados universitarios [15]. A pesar de su importancia, la enseñanza de lenguajes de programación es difícil, ya que implica la comprensión de conceptos teóricos, el uso práctico de la semántica del lenguaje, la codificación sintáctica y la lógica de razonamiento para la resolución de problemas [16-17]. Por este motivo el aprendizaje de la programación ha recibido mayor atención. Sin embargo, el enfoque tradicional del aprendizaje de la programación, el cual está basado en lenguajes textuales, para muchos estudiantes es demasiado difícil de aprender, lo que a menudo producen bajas tasas de éxito de aprendizaje en los primeros años [17],[19]. [16][17],[18],[17],[19].

Por otro lado, se debe tener en cuenta que el modelo educativo utilizado en las aulas en los grados de ingeniería suele ser: a) auditivo, b) abstracto, c) deductivo, d) pasivo y e) secuencial, en contraste con la mayoría de los estudiantes que son: a) visuales, b) sensibles, c) inductivos y d) activos [15]. Esta diferencia, asociada con la dificultad de aprender un lenguaje de programación, contribuye principalmente a la falta de interés del estudiante en los cursos de informática [18], el bajo rendimiento de los estudiantes y la frustración de los profesores [17]. Por tanto, existe un problema de desmotivación y rendimiento en los modelos actuales de aprendizaje de la programación, el cual se hace más visible en los cursos introductorios de dicha materia.

ueden

cursos introductorios de programación [20]: a) la experiencia previa del estudiante en programación obtenida de las escuelas secundarias; b) la experiencia con los conocimientos previos en otras ciencias en las que se emplea el ordenador; c) la relación entre los estilos de aprendizaje de los estudiantes y el modelo de aprendizaje del lenguaje de programación; d) las expectativas de los resultados del aprendizaje y la autoeficacia de los estudiantes; y e) las emociones experimentadas por los estudiantes [15]. Estos factores pueden influir de forma positiva o negativa en el proceso de aprendizaje, porque de estos depende que los contenidos acaben siendo significativos. Adicionalmente, los problemas en los resultados del aprendizaje también podría deberse en parte a una forma incorrecta de enseñar la programación [17], o al uso de un inadecuado material didáctico, ya que estos podrían no guiar al estudiante en el proceso de aprendizaje. Todos estos elementos condicionan el modelo de aprendizaje de la programación, en el cual, y de acuerdo con los planteamientos educativos tradicionales, los estudiantes tienen que escribir el programa directamente utilizando un lenguaje de programación. Esto puede llevar a que los estudiantes, que tienen poca experiencia en programación, se sientan frustrados fácilmente y disminuya la motivación de aprendizaje [21]. Además, para los maestros también lleva mucho tiempo y trabajo crear materiales de aprendizaje de programación textual usados habitualmente en los modelos educativos tradicionales [22].

Para solventar en parte estos problemas se han desarrollado muchas herramientas y entornos de programación para facilitar el aprendizaje y la comprensión eficiente [22],[23]. En este sentido las herramientas deben presentar los conceptos de programación más relevantes enseñados en un curso introductorio de programación, asegurándose de incluir los conceptos que los estudiantes consideran difíciles [17]. Según Moons [17] se dispone de cuatro grandes formas de enseñar programación. El primero es mediante el uso de una determinada metodología de educación en programación orientada a un cierto orden para introducir varios temas de programación. El segundo es mediante el empleo de diversas técnicas de aprendizaje activo inspiradas en el constructivismo, como el juego de roles, la narración activa de historias, los talleres, etc. El tercer enfoque es utilizar un lenguaje de programación que esté diseñado para estudiantes novatos en programación. El cuarto enfoque consiste en el uso de entornos de software. Moons identifica tres tipos de entornos de software para el aprendizaje de la programación [17]: a) entornos basados en micro mundos, como por ejemplo Scratch [24], LOGO [25], Karel the Robot [26], Jeroo [27], Alice [28], Kit Lego Mindstorms [29] y CMOTION [30]; b) herramientas de visualización de algoritmos, como por ejemplo Tango [31], Animal [32], Jawaa [33], Jhavé [34], Alvis Live! [35] y VisBack [36]; y c) herramientas de visualización y edición de código fuente, como DrJava [37], BlueJ [38], ProfessorJ [39], JGrasp [40], JIVE [41], JEliot 3 [42] y Ville [43]. De las herramientas listadas en los literales b y c la mayoría se orientan al paradigma de POO como es el caso de BlueJ que es ampliamente utilizado en los cursos de programación en Java. Hay que señalar que los lenguajes de orientación de objetos (como Java), en contraposición a otros lenguajes, conllevan una serie de conceptos adicionales como clases, llegar a ser difíciles de comprender, pero sin embargo tienen la ventaja de que hasta la fecha sigue siendo uno de los lenguajes más populares en los entornos profesionales, ya que ofrece la posibilidad de desarrollar aplicaciones para múltiples plataformas y tecnologías, lo cual puede llegar a ser motivador para los estudiantes que ven en su aprendizaje un aliciente profesional.

Las herramientas basadas en micromundos promueven un alto nivel de motivación y una percepción positiva de la programación de aprendizaje [44]. Estas herramientas facilitan la programación y el aprendizaje a través del diseño y la interfaz de programación visualizados, debido a que permiten a los usuarios programar manipulando elementos del programa gráficamente en lugar de especificarlos textualmente [45]. Estas características de los entornos gráficos permiten un proceso de aprendizaje práctico, basado en el concepto de aprender haciendo [22]. Con respecto a las herramientas para la visualización de algoritmos, los estudiantes que emplean estas herramientas disminuyen los errores semánticos y mantienen la atención en los detalles específicos del comportamiento del algoritmo [35]. Por último, las herramientas de visualización y edición de código fuente permiten ver a los estudiantes la estructura de los programas en tiempo de ejecución y tiempo de diseño (mediante artefactos como diagramas de clases), incluso en algunos casos permiten la generación automática de código [17]. Sin embargo, este tipo de herramientas tienen la limitación de que el estudiante se tiene que enfrentar con la complejidad sintáctica de utilizar un lenguaje de programación.

A diferencia de otras herramientas, la solución presentada en este artículo combina las características de los entornos basados en micro mundos con las herramientas de visualización de Algoritmos. Esta solución incorpora un mecanismo de *scaffolding* [46] donde se presenta al estudiante pequeños fragmentos de código fuente combinados con expresiones gráficas para que vaya relacionando las fases de solución de un problema con una sintaxis textual específica de un lenguaje concreto de programación. De tal forma que relaciona las expresiones sintácticas con expresiones gráficas según las capacidades del estudiante, haciendo que poco a poco el estudiante vaya dominando la sintaxis del lenguaje de programación que utiliza.

III. VILEP (VISUAL LEARNING OBJECT-ORIENTED PROGRAMMING)

En esta sección se describen los detalles de diseño e implementación de la herramienta, que ha sido desarrollada empleando técnicas de MDE (Model-driven engineering), que por medio de modelos específicos y una serie de transformaciones entre modelos se llega a una instancia concreta. El diseño se estructura en tres principales componentes: a) un metamodelo creado con Eclipse Modeling Framework (EMF), el cual permite modelar la construcción visual del programa que desarrolla el estudiante; b) una interfaz gráfica de usuario diseñada con Sirius para que pueda interactuar el estudiante con la herramienta de forma visual; y c) unas reglas de transformación de modelos para realizar la generación de código mediante Acceleo, a partir del programa que

onstruye el estudiante en el editor gráfico. El resultado es un *plug-in* que el estudiante puede instalar en Eclipse. Para lo cual debe agregar las carpetas con los metamodelos y editor gráfico de la herramienta, con lo cual se agrega una nueva opción para crear proyectos de VILEP en donde se pueden crear archivos con extensión ".javamodel" para que los estudiantes diseñen sus programas.

A. Definición del Metamodelo

Esta subsección describe el metamodelo diseñado para el dominio de la POO, el cual permite ignorar la complejidad de la sintaxis de un lenguaje. La Figura 1 muestra las metaclases del metamodelo y sus relaciones. Las metaclases utilizadas para describir las partes de un programa orientado a objetos son:

- *Explanation*: Esta metaclase representa el concepto de comentario en un código fuente y permite al estudiante insertar de forma automática una breve descripción del código que está creando a partir de su diseño.
- *Project*: Esta metaclase representa el concepto de proyecto que crea el estudiante.
- *Class*: Esta metaclase representa las clases que implementará el estudiante en su proyecto.
- *Method*: Esta metaclase representa los métodos o funciones que están en una clase.
- *Object*: Esta metaclase agrupa las metaclases hijas *Operator*, *Variable y Message*, la cuales representan respectivamente las operaciones matemáticas disponibles (suma, resta, multiplicación, división y resto), las declaraciones de las variables junto con los tipos de datos (int, float, double, String y boolean), y la lectura y salida de información por consola.
- *Argument*: Esta metaclase modela los argumentos de entrada o de salida de los métodos.

Estos componentes se relacionan entre sí: "*Project*" contiene "*Class*" y éste a su vez contiene "*Method*". Los métodos se relacionan con otros métodos y puede contener varios "*Object*" los cuales representan las operaciones matemáticas, lectura y escritura por teclado. Además, "*Method*" puede tener "*Argument*" cuyas referencias permiten a los métodos disponer de variables locales para sus operaciones.

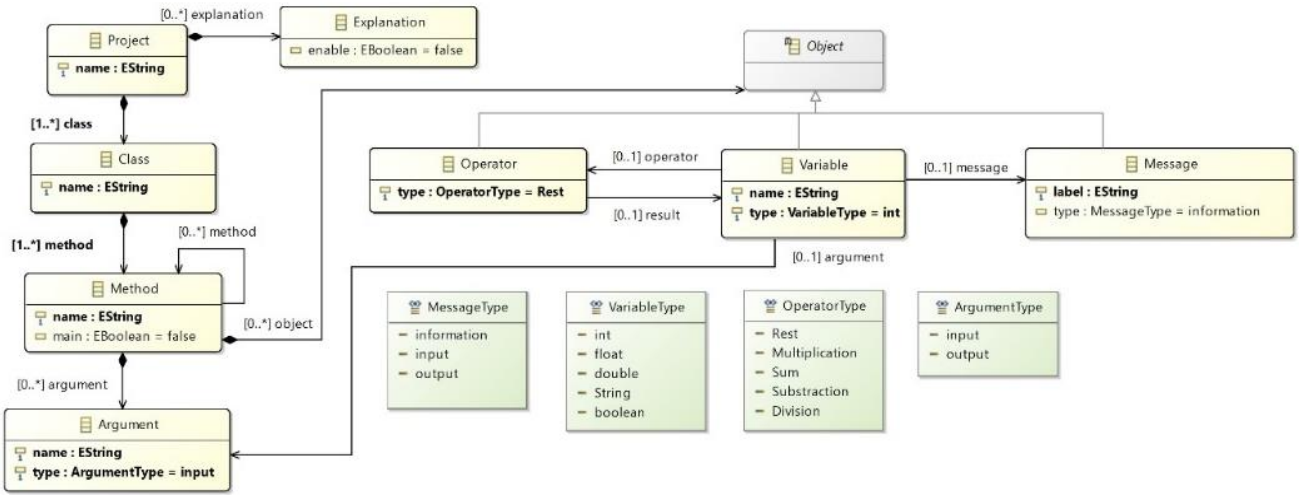

Fig. 1. Metamodelo propuesto para la herramienta visual

B. Transformación de modelos

Para la generación del código fuente en Acceleo se han implementado unas reglas de transformación que secuencian una serie de tareas básicas que se deben realizar para cualquier programa sencillo y que hemos especificado en el modelo de transformación. Se listan a continuación estas tareas para la creación de ficheros Java (.java):

1. Crear los ficheros del programa.
2. Declarar las librerías requeridas para los métodos.
3. Declarar las clases.
4. Declarar los métodos.
5. Declarar las variables empleadas.
6. Ingresar valores.
7. Realizar operaciones matemáticas.
8. Visualizar valores.

Estas reglas básicas que se aprenden al inicio de los cursos introductorios de programación son representadas de forma gráfica y delimitadas las relaciones que tiene cada uno de los componentes que acaban formando un programa. Por ejemplo, los métodos se crean solo dentro de las clases, las variables dentro de los métodos, las operaciones aritméticas reciben valores numéricos almacenados en variables y su resultado se asigna en otra variable. De esta forma los estudiantes se centran en el concepto de la secuencia de tareas que permiten resolver un problema, antes de enfrentarse a la sintaxis de un lenguaje. Con esta herramienta los estudiantes aprenden a realizar operaciones aritméticas, entrada y salida de datos por consola, uso de variables, declaración de métodos, tipos de datos y declaración de clases.

Las reglas de transformación que se han definido, y que transforman las expresiones gráficas del programa que diseña el estudiante, generan un código (en este caso en Java) que contiene: a) Llamado a la Clase *Scanner* del paquete *java.util* para el ingreso de datos por consola, b) Declaración de una instancia Scanner para almacenar la información ingresada por consola, c) Declaración de variables, d) Escritura en pantalla de mensajes con el método *println()*, e) Ingreso de valores con el método

nextLine() ejo de pantalla y almacenamiento en una variable, f) Declaración de operaciones, y g) Escritura en pantalla de los resultados de las operaciones con el método *println().*

IV. APRENDIZAJE VISUAL DE LA PROGRAMACIÓN CON VILEP

En esta sección se describe la interfaz de usuario de la herramienta, y su uso desde un enfoque docente. VILEP permite que el estudiante trabaje con conceptos básicos de POO, como son clases, métodos, argumentos, y también con otros conceptos de programación en general como son expresiones aritméticas, asignaciones, operaciones de entrada y de salida. En la Tabla 1 siguiente se resume algunas de las diferencias entre Eclipse y VILEP cuando se emplean en las primeras etapas de la enseñanza de la programación.

TABLA 1. PRINCIPALES DIFERENCIAS ENTRE ECLIPSE Y VILEP

Eclipse	VILEP
El código fuente debe escribirse de forma manual.	Generación de código fuente de forma automática
Los comentarios deben ser escritos por el programador.	Comentarios didácticos para la lectura e interpretación del código fuente.
Compilación y ejecución bien integrada.	Para la compilación y ejecución del programa se debe generar un nuevo documento.
Empleo de sintaxis textual para todos los conceptos de programación.	Empleo de sintaxis gráfica de los conceptos básicos de programación.
No dispone de un mecanismo de Scaffolding.	Mecanismo de Scaffolding que ajusta la sintaxis al nivel de conocimiento del programador.

Los conceptos básicos de programación que incorpora la herramienta son representados mediante componentes visuales disponibles en una paleta de controles, donde el estudiante puede seleccionar y arrastrar sobre el editor (denominado Lienzo) donde realiza el diseño de un programa (ver Figura 2). El editor visualiza la composición del programa en cada momento mostrando algunas partes de la sintaxis de Java y ocultando otras (las más complejas) mediante iconos visuales. Por ejemplo, la herramienta muestra la sintaxis completa de una instrucción sencilla en Java como es una operación de salida: "`System.out.println("Ingrese a:");`" (ver la Marca A de la Figura 2). Sin embargo, una instrucción más compleja como por ejemplo una operación de entrada sobre una variable como la siguiente: "`a=Integer.parseInt(leer.nextLine());`", es representada de forma visual (ver Marca B de la Figura 2), donde un símbolo en forma de lápiz representa que se trata de una operación de escritura del usuario por el teclado y un símbolo de una "x" entre corchetes expresa que el resultado se asigna a una variable (en este caso una variable denominada "a" de tipo *int*, ver Marca B de la Figura 2). Al no enfocarse la actividad del estudiante en la sintaxis del lenguaje, ésta presenta menor carga cognitiva, por lo que el estudiante puede centrarse más en los conceptos de programación durante la creación de programas. Sin embargo, la herramienta va eliminando niveles de abstracción y mostrando más detalle de las estructuras del la sintaxis. El profesor ajusta a cada problema o actividad diferentes niveles de scaffolding de acuerdo con diferentes niveles de conocimiento. Cada estudiante tiene asociado un perfil de conocimiento y la herramienta ajusta el scaffolding según su perfil. Según va avanzando el estduainte en las actividades se va ajustando su perfil. De esta forma, VILEP adapta el nivel de *scaffolding* para un entendimiento progresivo del estudiante en la creación de un programa para un lenguaje de programación concreto. La herramienta ofrece cuatro funcionalidades principales:

- Agregar al programa declaraciones de clases, métodos y variables. Estos componentes están en la paleta y el estudiante los arrastra para componer el programa (Figura 2).
- Realizar escritura y lectura por consola (operaciones de entrada y salida estándar).
- Realizar operaciones matemáticas básicas (suma, resta, multiplicación, división y resto). A estas operaciones se pueden asociar las variables con los valores a operar y la variable en la cual se asigna el resultado de la operación.
- Describir invocaciones de métodos y estructuras de control selectivas e iterativas

La Tabla 2 muestra los componentes de VILEP y su descripción. Los estudiantes pueden diseñar algoritmos básicos en forma gráfica sin entrar en mayores detalles de la sintaxis de un lenguaje. Los diseños gráficos que va generando el estudiante se interpretan con un flujo de lectura de arriba abajo y de izquierda a derecha, el cual lo traduce la herramienta y lo inserta en un archivo ".java". Sin embargo, no se han considerado algunos conceptos como la declaración de objetos o herencia, Teniendo en cuenta el significado de las representaciones visuales de la Tabla 1, se puede interpretar fácilmente el fragmento de programa de la Figura 2: se declara una clase denominada Operaciones, la cual contiene el método "Main". Dentro de este método se describe la siguiente secuencia de instrucciones: se escribe un mensaje por pantalla ("Ingresar A"), se declara y lee el teclado en la variable "a", posteriormente realiza lo mismo para la variable "b" y a continuación realiza la multiplicación de sus dos contenidos y el resultado lo asigna a la variable "x" declarada como *int*, luego la variable "c", que fue ingresada por teclado, se suma a "x" y se asigna el resultado en "y", por último se imprime el contenido de esta variable por pantalla.

TABLA 2. PRINCIPALES COMPONENTES Y SUS REPRESENTACIONES VISUALES

Componente	Representación visual	Descripción
Class		Declaración de una clase
Method		Declaración de un método
Argument		Parámetros reales de métodos
Variable		Declaración de variable
Operador		Operador aritmético
Message		Escritura de texto por consola
Input		Lectura desde el teclado
Output		Escritura de una variable por consola
Connection		Enlace entre componentes del programa

A. *Método docente*

El objetivo de la herramienta es reducir la carga de trabajo cognitiva de un lenguaje para los estudiantes inexpertos en programación. Con este objetivo, las actividades de aprendizaje con la herramienta se desarrollan con tareas cortas de trabajo, de esta forma el estudiante va adquiriendo confianza para enfrentarse a problemas de mayor dificultad, los cuales requerirán estructuras del lenguaje más complicadas. El enfoque docente se basa principalmente en que los estudiantes usarán la herramienta para aprender los principios de la sintaxis básica del lenguaje de programación. Mientras emplean el editor de la herramienta, visualmente van observando cómo se van construyendo algunas líneas de código y al final observarán el código completo. Esto permite al estudiante ver inmediatamente cómo las decisiones que toma en el diseño de los programas se reflejan directamente en una sintaxis específica de un lenguaje de programación. En este contexto, el aprendizaje visual que se genera permitirá que los estudiantes poco a poco ganen confianza en la escritura de los programas para un lenguaje concreto.

La metodología docente de uso de la herramienta VILEP se resume a grandes rasgos en los siguientes pasos:

1. El profesor explica los conceptos de programación básicos y presenta brevemente la sintaxis de Java. Además, debe mostrar la representación visual de los componentes de VILEP de estos conceptos.
2. El profesor propondrá un enunciado de un problema a resolver y reflexionará con los estudiantes sobre los pasos a dar para resolverlo.
3. Posteriormente, los estudiantes usan VILEP de forma autónoma para implementar el programa que resuelve el problema propuesto. Durante este proceso de diseño la herramienta irá mostrando fragmentos de código fuente asociado a determinadas acciones, y ocultando otras. Finalmente, la herramienta generará el código fuente que ha implementado ampliado con comentarios aclaratorios.
4. El profesor explicará las dudas y los estudiantes revisarán y podrán ejecutar el programa que han obtenido y verificar la validez de sus soluciones.

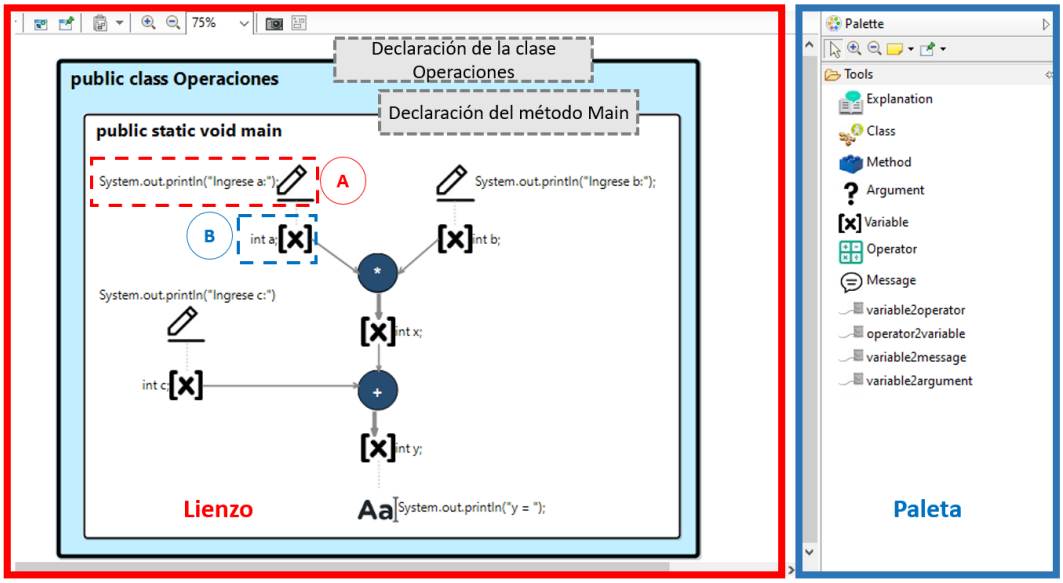

Fig. 2. Composición de un programa con VILEP

V. EXPERIENCIA EN EL AULA Y RESULTADOS

Para demostrar la validez de la herramienta en el proceso de enseñanza se realizó un estudio exploratorio con estudiantes en el aula. Se detalla a continuación la misma.

A. *Método docente*

La experiencia con estudiantes tuvo como objetivo validar si la utilización de VILEP en un curso de introducción a la programación mejora los resultados de aprendizaje y el estado emocional del estudiante durante el proceso de aprendizaje.

B. *Muestra*

La muestra seleccionada fueron estudiantes del primer curso del grado de Electrónica y Automatización, de la Universidad de las Fuerzas Armadas ESPE, en Quito Ecuador, en el segundo semestre de 2018, estando constituida por 19 sujetos (17 hom

cuales no tenían experiencia previa en programación. Esta muestra se organizó de forma aleatoria en dos grupos: grupo experimental (GE) y grupo de control (GC).

C. *Variables e instrumentos*

La variable independiente fue la herramienta docente aplicada: en el GC se aplicó la herramienta docente clásica de un entorno de desarrollo, mientras que en el GE se usó la herramienta VILEP. Las variables dependientes que se midieron fueron el nivel de conocimiento adquirido y las emociones positivas y negativas experimentadas. Para ello, en los dos grupos, se hizo un pre-test de estas variables al inicio de la experiencia, y un pos-test al finalizar. Los instrumentos para medir estas variables fueron dos escalas. En primer lugar, una escala de conocimiento con 6 ítems multi-opción diseñada específicamente para la experiencia. Esta escala planteaba cuestiones sobre conceptos básicos de POO en las que el estudiante tenía que interpretar código

validada para medir las emociones: PANAS (Positive Affect & Negative Affect Scale) de Watson [47]. El motivo de usar esta escala es que ya está validada en el contexto educativo y permite valorar las emociones positivas y negativas del estudiante en la tarea de aprendizaje. La escala se compone de 20 términos (Tabla 3) que describen emociones de carácter positivo o negativo (10 de ellas positivas y 10 negativas). El sujeto debe valorar cómo se siente para cada uno de estos términos emocionales mediante una escala Likert con 5 opciones de respuesta (nada, muy poco, algo, bastante, mucho).

TABLA 3. ESCALA DE EMOCIONES PANAS

Términos de emociones positivas		**Términos de emociones negativas**	
Interesado	Decidido	Disgustado/enfadado	Tenso
Dispuesto	Atento	Culpable	Avergonzado
Animado	Activo	Temeroso	Nervioso
Entusiasmado	Enérgico	Enojado	Intranquilo
Orgulloso	Inspirado	Irritado	Asustado

D. Método

La ejecución de la experiencia se realizó con los estudiantes del curso de Introducción a la Ingeniería, en el que se contempla en uno de los resultados de aprendizaje estudiar los conceptos básicos de la programación. Para lo cual se empleó 2 de las 6 sesiones del curso destinadas para alcanzar el resultado del aprendizaje. La experiencia comenzó con una explicación a los estudiantes de los objetivos de la actividad y solicitando el consentimiento de participación (participaron el 100% de los estudiantes). A continuación, se organizaron los participantes aleatoriamente en dos grupos: a) GE (Grupo Experimental), grupo constituido por 10 participantes que utilizaron la herramienta VILEP, y b) GC (Grupo de Control), constituido por 9 participantes que tuvieron como método docente el habitual utilizando el entorno de Eclipse. Aun cuando la herramienta se empleó de forma exploratoria para esta experiencia de algunos conceptos básicos de POO, pero se podría utilizar para el resto de los conceptos del curso.

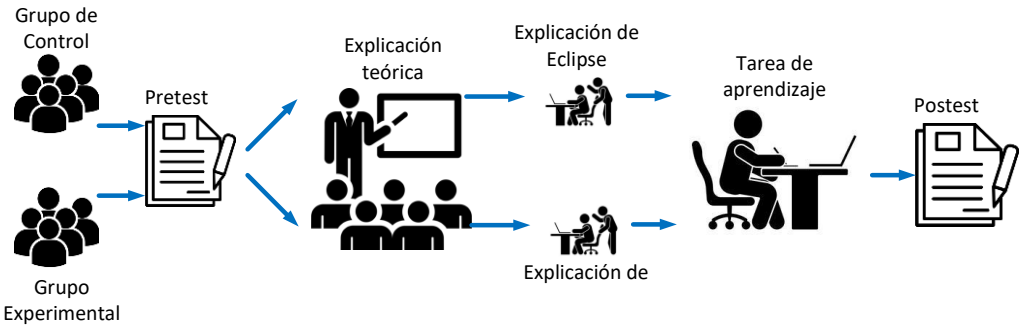

Fig. 3. Metodología de la experiencia en el aula

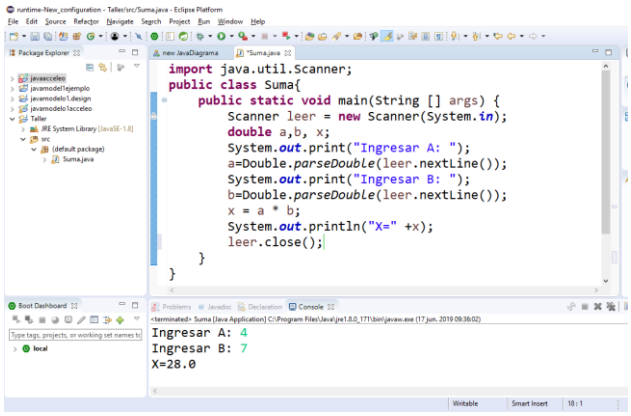

Fig. 4. Tarea de aprendizaje del GC

La Figura 3 muestra el desarrollo de la experiencia realizada. Una vez constituidos los grupos comenzó la intervención realizándose una evaluación inicial del conocimiento y del estado emocional de los estudiantes. A continuación, el profesor (fue el mismo en los dos grupos) explicó los fundamentos teóricos de POO (clases, métodos y atributos) y la sintaxis básica de Java (declaraciones de clases y atributos, operadores aritméticos, entrada y salida), empleando la herramienta específica para cada grupo. Posteriormente en el GE empleó la herramienta VILEP y el GC Eclipse para desarrollar los primeros programas con asesoramiento del profesor. A continuación, ambos grupos hicieron varias pruebas implementando un programa en Java muy básico, el GE usando VILEP mediante programación visual y el GC usando Eclipse con programación clásica textual. La Figura 4 muestra una captura de un programa que desarrollaron estudiantes del GC con Eclipse, mientras que en la Figura 2 se puede ver un programa desarrollado por estudiantes del GE con VILEP. Por último, se volvió a evaluar el conocimiento y las emociones tras la realización de la tarea. La planificación completa y su distribución temporal para cada grupo se muestran en la Tabla 4.

E. Resultados experimentales

Para la comparación de los resultados obtenidos en ambos grupos se realizó dos análisis: a) un contraste de hipótesis de igualdad de medias de las variables medidas y b) una comparación de las tasas de errores en la tarea de aprendizaje. Para realizar el contraste de igualdad de medias se ha definido las siguientes variables estadísticas a estudio:

- CONCIMIENTO_PRE: Son los resultados del nivel de conocimiento al inicio de la experiencia.
- CONCIMIENTO_POS: Es el conocimiento de los estudiantes después de la experiencia.
- EMOCIONES_POSITIVAS_PRE: Son las emociones positivas de los estudiantes al inicio de la experiencia.
- EMOCIONES_POSITIVAS_POS: Muestra las emociones positivas de los estudiantes después de la experiencia.

- EMOCIONES_NEGATIVAS_PRE: Mide las emociones negativas de los estudiantes al comenzar la experiencia.
- EMOCIONES_NEGATIVAS_POS: Son las emociones negativas de los estudiantes al finalizar la experiencia.
- PERCEPCIONES_NEGATIVAS_TAR: Mide las percepciones negativas de los estudiantes una vez realizada la experiencia.
- PERCEPCIONES_POSITIVAS_TAR: Muestra las percepciones positivas de los estudiantes sobre la experiencia.

TABLA 4. TEMPORALIDAD DE LA EXPERIENCIA

Fase	Grupo Experimental	Grupo Control
Presentación y formación de grupos	15”	15”
Realización pre-test (conocimiento y emociones)	30”	30”
Explicación conceptos POO y Java con VILEP	1h	-
Explicación conceptos POO y Java con Eclipse	-	1h
Empleo de VILEP con asistencia del profesor	30”	-
Empleo de Eclipse con asistencia del profesor	-	30”
Realización de tareas autónomas de implementación programas con VILEP	30”	-
Realización de tareas autónomas de implementación programas con Eclipse	-	30”
Realización post-test (conocimiento y emociones)	30”	30”
TIEMPO DE LA EXPERIENCIA	**3h 15”**	**3h 15”**

La Tabla 5 muestra la estadística descriptiva comparando ambos grupos. En la Tabla 4 se observa que el nivel de conocimiento al terminar la experiencia es mayor en el grupo que usó VILEP frente a los que no lo usaron (variable CONOCIMIENTO_POS=3,8 vs. 3,67). De forma similar también se observa en esta misma tabla que las emociones positivas fueron mayores al acabar la experiencia en los estudiantes con VILEP que en los estudiantes que usaron Eclipse (variable EMOCIONES_POSITIVAS_POS=38,8 vs. 37,78). Sin embargo, también se identificó que las emociones negativas fueron ligeramente mayores en el grupo experimental que en el de control. Con el objetivo de constatar si estas diferencias son significativas o no estadísticamente se realiza un contraste de hipótesis de igualdad de medias. Para ello, en primer lugar, se identificó qué variables seguían una distribución normal, aplicando el test *Shapiro Wilk* ($p=0.05$) al tratarse de muestras pequeñas.

En este estudio se obtuvo que todas seguían una distribución normal excepto las variables EMOCIONES_POSITIVAS_POS/y/PERCEPCIONES_POSITIVAS_TAR. Para estas dos variables se realizó el test de contraste de comparación de medias mediante la prueba no paramétrica de *Mann-Whitney*, mientras que para el resto de las variables se realizó la prueba de *t-Student* (ambas con un nivel de significancia $p=0.05$). El contraste de hipótesis en ambas pruebas indicó que no había diferencia entre las medias en ninguna de las variables. Por tanto, aunque no se ha hallado una diferencia significativa, sí que se puede afirmar que hay una tendencia de mejora en el nivel de conocimiento y en las emociones positivas del uso de VILEP frente al uso de Eclipse, si bien hay una tendencia de empeoramiento en las emociones negativas.

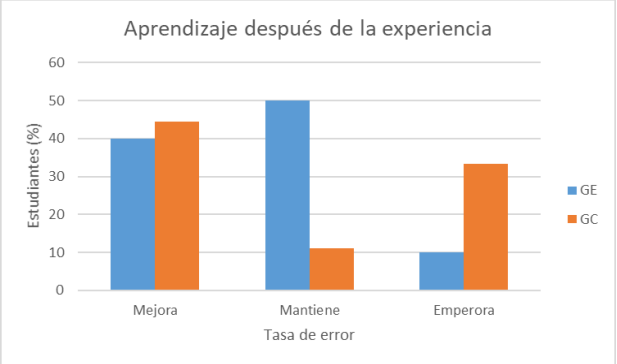

Fig. 5. Resultados de aprendizaje según la tasa de errores

Por otro lado, analizando el número de estudiantes que aumentaron sus emociones positivas en un grupo y en otro, se observa que en el grupo que usó VILEP, el 60% de los estudiantes aumentaron sus emociones positivas durante la realización de la tarea, mientras que en el grupo de control solo fueron el 44,4% de los estudiantes. Por tanto, se puede entender que el uso de la programación visual con VILEP fomenta las emociones positivas de algunos estudiantes en el proceso de aprendizaje de la programación, aunque no hay que descuidar que el uso de la herramienta no disminuyó las emociones negativas tanto como ocurrió en el grupo de control (40% de GE frente 66,6% del GC). Estos resultados podrían estar relacionado con el efecto colateral de la carga cognitiva que conlleva el uso de representaciones visuales en el aprendizaje de la programación [48].

TABLA 5. ESTADÍSTICOS DESCRIPTIVOS DE LOS DOS GRUPOS

	Grupo experimental (GE) (*N=10*)				Grupo de control (GC) (N=9)			
	Mínimo	**Máximo**	**Media**	**Desviación estándar**	**Mínimo**	**Máximo**	**Media**	**Desviación estándar**
CONOCIMIENTO_PRE	1	6	3,50	1,581	0	6	2,89	1764
CONOCIMIENTO_POS	2	6	3,80	1,549	1	6	3,67	1658
EMOCIONES_POSITIVAS_PRE	33	46	38,90	3,814	28	44	37,22	5,019
EMOCIONES_POSITIVAS_POS	28	44	38,80	4,417	29	47	37,78	6360
EMOCIONES_NEGATIVAS_PRE	10	22	14,50	4,275	10	22	14,67	3841
EMOCIONES_NEGATIVAS_POS	10	22	13,70	4,029	10	17	12,89	2,619
PERCEPCIONES_NEGATIVAS_TAR	1,2	2,8	2,060	,5337	1,2	3	1911	,5754
PERCEPCIONES_POSITIVAS_TAR	4	5	4,70	,483	4	5	4,78	,441

Como ya se ha indicado, en una segunda fase de análisis de los resultados, se ha revisado las tasas de error cometidas en el pre-test y post-test para cada grupo. En el GE al finalizar la experiencia el 40% de los estudiantes mejoró sus resultados, el 50% la mantuvo y el 10% de los estudiantes incrementaron sus errores. En el GC el 44.5% de los estudiantes mejoraron sus resultados, el 11.1% mantuvo el mismo nivel y el 33.4% aumentó sus errores. Como se observa, casi la mitad de los estudiantes de los dos grupos mejoraron sus resultados (44,5% del GC vs. 40% del GE) sin embargo, en el grupo de control existió un incremento significativo de estudiantes con más errores frente al experimental (33,4% del GC vs. 10% del GE). La Figura 5 muestra gráficamente estos datos, donde se puede ver cómo

resultados o bien mantuvieron sus resultados, y muy pocos la empeoraron, frente a los que usaron Eclipse, en cuyo grupo hubo un elevado número de estudiantes que cometieron más errores en el post-test. Es decir, a tenor de estos datos parece ser que el uso de la programación visual con la herramienta VILEP hace que los estudiantes cometan menos errores en la codificación de programas. Los autores piensan que esto podría ser debido a que la programación visual con VILEP permite que el estudiante se centre únicamente en los conceptos y estructuras de programación durante el proceso de aprendizaje, haciendo que posteriormente, al aplicarlos en la codificación de programas, se cometan menos errores. Esto explicaría que los estudiantes que usaron Eclipse cometiesen más errores de codificación ya que en el proceso de aprendizaje el estudiante tenía que trabajar conjuntamente los conceptos y estructuras de programación junto con la dificultad de la sintaxis del lenguaje.

VI. Conclusiones

En los grados ingeniería de informática y afines, las asignaturas de introducción a la programación presentan bajos resultados de aprendizaje y bajos porcentajes de aprobados. Uno de los motivos que genera estos malos resultados es la dificultad que tienen estos estudiantes inexpertos en entender y manejar adecuadamente la sintaxis de los lenguajes de programación [12]. Estas dificultades acaban generando una desmotivación en el estudiante, debido a que no sólo debe enfrentarse al reto de entender conceptos fundamentales de programación, sino que además debe aprender el lenguaje de programación asociado a esos conceptos. En este artículo se presenta la herramienta VILEP, la cual facilita un editor de programación visual que abstrae al estudiante de la complejidad del uso del lenguaje y se centra en los conceptos de programación. La herramienta implementa un mecanismo de Scaffolding que permite al estudiante visualizar o no, las partes más complejas de las estructuras sintácticas de los lenguajes de programación. El mecanismo agrega representaciones visuales y las combina con fragmentos de código fuente, de tal forma que a medida que el estudiante va asimilando la sintaxis del lenguaje la herramienta va disminuyendo el nivel de abstracción y mostrando más detalle de las estructuras sintácticas hasta que el estudiante termina desarrollando el código fuente directamente. Se estructuró este primer estudio exploratorio que contó con los estudiantes de primer curso del grado de Electrónica y Automatización organizándose dos grupos de trabajo para desarrollar un programa en Java: uno de ellos usó la herramienta VILEP propuesta y el otro trabajó con el entorno de desarrollo Eclipse mediante programación textual (herramienta habitual usada en las asignaturas de informática). Se utilizó una escala de conocimiento para medir los resultados de aprendizaje y la escala PANAS para medir las emociones de los estudiantes durante el proceso de aprendizaje. Se puede concluir que la herramienta de programación visual propuesta redujo considerablemente el número de estudiantes que al terminar la experiencia cometían errores con el lenguaje de programación respecto

iantes

menos). Además, se halló que hubo más estudiantes que experimentaron emociones positivas durante la tarea de programación cuando usaban la herramienta VILEP (60% de los estudiantes del grupo) que cuando no la usaban (44,4% del grupo).

Al tenor de estos resultados se abren algunas líneas de trabajos futuros. El análisis de los resultados es un análisis exploratorio por lo que es necesario replicar nuevas experiencias en otras universidades con un análisis estadístico más profundo y con un mayor número de participantes, que permita confirmar si las mejoras en los resultados de aprendizaje y en las emociones halladas son estadísticamente significativas. Además, se deben realizar estudios que analicen las correlaciones entre los resultados de aprendizaje y las emociones durante el proceso de aprendizaje de la programación. Por otro lado, se debe incorporar nuevas funcionalidades como ingeniería inversa para representar de forma gráfica programas realizados de forma textual, ampliación de funciones para POO y recursos web del proyecto con información adicional para que la herramienta pueda ser empleada por otros usuarios.

Agradecimientos

Este trabajo ha sido financiado gracias a iProg del MINECO (ref. TIN2015-66731-C2-1-R) y e-Madrid-CM (ref. P2018/TCS-4307) con fondos FSE y FEDER.

Referencias

[1] H. Amer and A. Ain, “Smart – Learning Course Transformation for an Introductory Programming Course,” *2017 IEEE 17th Int. Conf. Adv. Learn. Technol.*, pp. 463–465, 2017.

[2] A. Forte and M. Guzdial, “Motivation and Nonmajors in Computer Science : Identifying Discrete Audiences for Introductory Courses,” vol. 48, no. 2, pp. 248–253, 2005.

[3] L. Mclver and D. Conway, “Seven Deadly Sins of Introducctory Prograimmting Language Design Linda,” *Notes Queries*, pp. 309–316, 1996.

[4] E. Kaila, M. Laakso, T. Rajala, A. Mäkeläinen, and E. Lokkila, “Technology-Enhanced Programming Courses for Upper Secondary School Students,” pp. 683–688, 2018.

[5] R. Mahmudur and R. Paudel, “Preliminary Experience and Learning Outcomes by Infusing Interactive and Active Learning to Teach an Introductory Programming Course in Python,” 2018.

[6] C. C. W. Hulls, A. J. Neale, B. N. Komalo, V. Petrov, and D. J. Brush, “Interactive Online Tutorial Assistance for a First Programming Course,” vol. 48, no. 4, pp. 719–728, 2005.

[7] M. Schmidt, V. Benzing, A. Wallman-Jones, M. F. Mavilidi, D. R. Lubans, and F. Paas, “Embodied learning in the classroom: Effects on primary school children’s attention and foreign language vocabulary learning,” *Psychol. Sport Exerc.*, vol. 43, pp. 45–54, 2019.

[8] O. Debdi, M. Paredes-Velasco, and J. A. Velazquez-Iturbide, “Influence of Pedagogic Approaches and Learning Styles on Motivation and Educational Efficiency of Computer Science Students,” *Rev. Iberoam. Tecnol. del Aprendiz.*, vol. 11, no. 3, pp. 213–218, 2016.

[9] K. Willey and A. Gardner, “Collaborative learning frameworks to promote a positive learning culture,” *Proc. - Front. Educ. Conf. FIE*, pp. 1–6, 2012.

[10] L. P. Nelson and M. L. Crow, “Do Active-Learning Strategies Improve Students’ Critical Thinking?,” *High. Educ. Stud.*, vol. 4, no. 2, pp. 77–90, 2014.

[11] X. Basogain-Olabe, M. Á. Olabe-Basogain, and J. C. Olabe-Basogain, “Pensamiento Computacional a través de la Programación: Paradigma de Aprendizaje,” *Rev. Educ. a Distancia*, vol. 46, no. 46, 2015.

[12] O. Aktunc, “A Teaching Methodology for Introductory

Programming Courses using Alice," *Int. J. Mod. Eng. Res.*, vol. 3, no. 1, pp. 350–353, 2013.
[13] C. Dumitrescu, R. L. Olteanu, L. M. Gorghiu, and G. Gorghiu, "Using virtual experiments in the teaching process," vol. 1, no. 1, pp. 776–779, 2009.
[14] M. Kölling, "The Greenfoot Programming Environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–21, 2010.
[15] J. van Niekerk and P. Webb, "The effectiveness of brain-compatible blended learning material in the teaching of programming logic," *Comput. Educ.*, vol. 103, pp. 16–27, 2016.
[16] G. M. M. Bashir and A. S. M. L. Hoque, "An effective learning and teaching model for programming languages," *J. Comput. Educ.*, vol. 3, no. 4, pp. 413–437, 2016.
[17] J. Moons and C. De Backer, "The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism," *Comput. Educ.*, vol. 60, no. 1, pp. 368–384, 2013.
[18] M. Koorsse, C. Cilliers, and A. Calitz, "Programming assistance tools to support the learning of IT programming in South African secondary schools," *Comput. Educ.*, vol. 82, pp. 162–178, 2015.
[19] P. Tshering, D. Lhamo, L. Yu, and A. Berglund, "How Do First Year Students Learn C Programming in Bhutan?," *Proc. - 5th Int. Conf. Learn. Teach. Comput. Eng. LaTiCE 2017*, pp. 25–29, 2017.
[20] C. J. Olelewe and E. E. Agomuo, "Effects of B-learning and F2F learning environments on students' achievement in QBASIC programming," *Comput. Educ.*, vol. 103, pp. 76–86, 2016.
[21] K. J. Harms, J. Chen, and C. Kelleher, "Distractors in parsons problems decrease learning efficiency for young novice programmers," *ICER 2016 - Proc. 2016 ACM Conf. Int. Comput. Educ. Res.*, pp. 241–250, 2016.
[22] J. M. Su and T. W. Lin, "Building a Simulated Blockly-Arduino-Based Programming Learning Tool: A Preliminary Study," *Proc. - 2018 7th Int. Congr. Adv. Appl. Informatics, IIAI-AAI 2018*, pp. 378–381, 2018.
[23] D. Saito, H. Washizaki, and Y. Fukazawa, "Work in progress: A comparison of programming way: Illustration-based programming and text-based programming," *Proc. 2015 IEEE Int. Conf. Teaching, Assess. Learn. Eng. TALE 2015*, no. December, pp. 220–223, 2016.
[24] S. Uludag, M. Karakus, and S. W. Turner, "Implementing IT0/CS0 with scratch, app inventor for android, and lego mindstorms," *SIGITE'11 - Proc. 2011 ACM Spec. Interes. Gr. Inf. Technol. Educ. Conf.*, pp. 183–189, 2011.
[25] D. M. Kurland and R. D. Pea, "Children's Mental Models of Recursive Logo Programs," *J. Educ. Comput. Res.*, vol. 1, no. 2, pp. 235–243, 1985.
[26] K. Dai, Y. Zhao, and R. Chen, "Research and practice on constructing the course of programming language," *Proc. - 10th IEEE Int. Conf. Comput. Inf. Technol. CIT-2010, 7th IEEE Int. Conf. Embed. Softw. Syst. ICESS-2010, ScalCom-2010*, no. Cit, pp. 2033–2038, 2010.
[27] B. Dorn, "Jeroo : A Tool for Introducing Object-Oriented Programming Northwest Missouri State University," *Control*, pp. 201–204, 2003.
[28] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, pp. 191–195, 2003.
[29] S. H. Kim and J. W. Jeon, "Introduction for freshmen to embedded systems using LEGO mindstorms," *IEEE Trans. Educ.*, vol. 52, no. 1, pp. 99–108, 2009.
[30] S. L. Finkelstein, A. Nickel, L. Harrison, E. A. Suma, and T. Barnes, "cMotion: A new game design to teach emotion recognition and programming logic to children using virtual humans," *Proc. - IEEE Virtual Real.*, pp. 249–250, 2009.
[31] J. T. Stasko, "Tango: A Framework and System for Algorithm Animation," *Computer (Long. Beach. Calif).*, vol. 23, no. 9, pp. 27–39, 1990.
[32] G. Rbling, M. Sch, B. Freisleben, and D.- Siegen, "The ANIMAL Algorithm Animation Tool," pp. 37–40, 2000.
[33] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S. H. Rodger, "JAWAA: Easy web-based animation from CS 0 to advanced CS courses," *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, pp. 162–166, 2003.
[34] T. L. Naps, J. R. Eagan, and L. L. Norton, "JHAVE - an environment to actively engage students in Web-based algorithm visualizations," *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, pp. 109–113, 2000.
[35] C. D. Hundhausen and J. L. Brown, "Designing, visualizing, and discussing algorithms within a CS 1 studio experience: An empirical study," *Comput. Educ.*, vol. 50, no. 1, pp. 301–326, 2008.
[36] C. Lacave, J. Á. Velázquez-Iturbide, M. Paredes-Velasco, and A. I. Molina, "Analyzing the influence of a visualization system on students' emotions: An empirical case study," *Comput. Educ.*, vol. 149, no. January, 2020.
[37] E. Allen and B. Stoler, "Dr Java : A lightweight pedagogic environment for Java," 2002.
[38] M. C. Jadud, "A First Look at Novice Compilation Behaviour Using BlueJ," *Comput. Sci. Educ.*, vol. 15, no. 1, pp. 25–40, 2005.
[39] K. E. Gray and M. Flatt, "ProfessorJ : AA gradual introduction to Java through language levels," *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl. OOPSLA*, pp. 170–177, 2003.
[40] J. H. Cross and D. Hendrix, "Workshop jGRASP: An integrated development environment with visualizations for teaching Java in CS1, CS2, and beyond," *Proc. - Front. Educ. Conf. FIE*, p. 1, 2006.
[41] P. V. Gestwicki and B. Jayaraman, "JIVE: Java interactive visualization environment," *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl. OOPSLA*, pp. 226–227, 2004.
[42] A. Moreno and M. S. Joy, "Jeliot 3 in a Demanding Educational Setting," *Electron. Notes Theor. Comput. Sci.*, vol. 178, pp. 51–59, 2007.
[43] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, "VILLE -- A Language-Independent Program Visualization Tool," *Seventh Balt. Sea Conf. Comput. Educ. Res. (Koli Call. 2007)*, vol. 88, no. January 2016, pp. 151–159, 2007.
[44] B. Kaučič and T. Asič, "Improving introductory programming with Scratch?," *MIPRO 2011 - 34th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. - Proc.*, pp. 1095–1100, 2011.
[45] J. M. Su and S. J. Wang, "A Web-Based learning activity integrated with scratch tool to support programming learning," *Ubi-Media 2017 - Proc. 10th Int. Conf. Ubi-Media Comput. Work. with 4th Int. Work. Adv. E-Learning 1st Int. Work. Multimed. IoT Networks, Syst. Appl.*, pp. 1–4, 2017.
[46] C. H. Chen and V. Law, "Scaffolding individual and collaborative game-based learning in learning performance and intrinsic motivation," *Comput. Human Behav.*, vol. 55, pp. 1201–1212, 2016.
[47] A. Watson, D., Clark, L., & Tellegen, "Development and validation of brief measures of positive and negative affect: the PANAS scales," *J. Pers. Soc. Psychol.*, vol. 54, no. 6, pp. 1063–1070, 1988.
[48] P. Crescenzi, A. Malizia, M. C. Verri, P. Diaz, and I. Aedo, "On Two Collateral Effects of Using Algorithm Visualizations," *Br. J. Educ. Technol.*, vol. 42, no. 6, pp. 145–147, 2011.

Darwin Alulema recibió los títulos de Ingeniero en Electrónica y Telecomunicaciones (Ecuador-2006), Abogado de los Juzgados y Tribunales de la República (Ecuador-2011), Máster en Teleinformática y Redes de Computadoras (Ecuador-2009), Máster en Aplicaciones Multimedia (España-2015) y Máster en Derecho Civil y Procesal Civil (Ecuador-2016). Desde el año 2007 es Profesor Titular la Universidad de las Fuerzas Armadas, en Ecuador.

Maximiliano Paredes-Velasco recibió el grado de Ph.D. en informática por la Universidad de Castilla-La Mancha, España, en 2006. Actualmente es Profesor de la Universidad Rey Juan Carlos. Sus intereses de investigación incluyen diferentes campos de aprendizaje apoyados por computadora (aprendizaje colaborativo, aprendizaje activo y aprendizaje móvil) e interacción humano-computadora. Es autor de numerosos artículos y varios documentos de conferencias internacionales. Su investigación se centra en el aprendizaje colaborativo, la computación ubicua y la interacción humano-computadora.