A Support Tool designed as Didactic Material for Teaching and Learning Programming

María Julia Blas, Federico Hauque, Santiago Re, Marta Castellaro

***Abstract*— Usually, programing courses evaluations use a comparative method to test their students. That is, the student must analyze the content of some piece of code in order to understand its logic. However, most courses do not use special support mechanisms for this task. This paper presents a software tool designed to support teaching and learning tasks related with programming courses. The main objective is to increase the motivation of the students using tools developed by their classmates. In future, the final software tool will be included as part of the course materials. Then, other students will have the opportunity to use the software to test their programming knowledge.**

***Index Terms*— Algorithms and data structures, programming, learning support tool, syntax errors.**

I. Introducción

COMO parte de las habilidades a desarrollar a nivel universitario, la educación en ciencias de la computación requiere que el alumno adquiera conocimientos de programación. La programación es una habilidad no sencilla de adquirir, la cual se perfecciona con la práctica (siendo más efectiva cuando dicha práctica es auto-dirigida) [1].

Uno de los principales problemas que enfrentan los docentes al momento de iniciar un curso, es lograr una adecuada motivación en los estudiantes a fin de generarles un compromiso con el proceso de aprendizaje [2]. Múltiples autores han trabajado sobre esta problemática, generando diferentes enfoques relacionados con estrategias de enseñanza y/o herramientas de soporte al aprendizaje [3], [4], [5]. Incluso diferentes formas de programación han sido propuestas, entre las que se destacan programación por demostración [6], [7]; programación visual [8]; programación mediante ejemplos [9]; programación gráfica [10] y programación física [11].

La cátedra Algoritmos y Estructuras de Datos (AEDD) de la carrera Ingeniería en Sistemas de Información de la Facultad Regional Santa Fe - Universidad Tecnológica Nacional, introduce al alumno en los conceptos básicos de programación a fin de presentar el paradigma procedural como primer acercamiento a dicha actividad. La especificación de algoritmos en C++ junto con el análisis de su correctitud y eficiencia, en combinación con el diseño de estructuras de datos básicas (tanto simples como compuestas), son algunos de los principales ejes que se desarrollan a lo largo del cursado. De esta manera, el principal objetivo de la cátedra es introducir al alumno en la programación y en el diseño de estructuras de datos que le permitan formular soluciones aplicables a distintos tipos de problemas dados. Como parte de los objetivos actitudinales, se espera que el alumno:

1) Desarrolle la capacidad de abstracción.
2) Ejercite el razonamiento lógico.
3) Consiga un análisis reflexivo y crítico ante diferentes tipos de soluciones aplicables a un mismo problema.
4) Tome decisiones basadas en las posibilidades y limitaciones de situaciones particulares.
5) Adquiera autonomía para resolver situaciones de forma metódica.
6) Realice un aprendizaje colaborativo.
7) Ejercite destrezas de comunicación (análisis conjunto, presentación y discusión de modelos y estrategias, lecturas de programas ya diseñados, entre otros).
8) Adquiera hábitos de predisposición al autoestudio (especialmente relacionados al empleo de distintos lenguajes de programación).
9) Incorpore el uso de tecnologías web como alternativas para la autogestión de información (como ser tutoriales, enlaces web y correo electrónico).

Para lograr cumplir con estos objetivos, la cátedra propone un conjunto de actividades a realizar durante el ciclo lectivo a modo de evaluación continua. Entre estas actividades se destacan cuestionarios implementados sobre la plataforma web de la materia [12], trabajos prácticos integradores a resolver en grupos de tres integrantes, resolución de ejercicios utilizando el juez en línea URI [13] y exámenes parciales a resolver tanto en papel como en computadora haciendo uso del IDE ZinjaI [14]. Ya sea como parte de los exámenes parciales o bajo la forma de cuestionarios, es usual que el alumno disponga de un conjunto de códigos fuente a ser analizados. Tales códigos suelen ser de baja complejidad debido a que el soporte brindado por los formularios web no posibilita el manejo de diferentes tipos de ejercicios y/o de ejemplos extensos. Además, la falta de soporte para la automatización de la corrección de este tipo de actividades, dificulta la incorporación de nuevos ejercicios.

María Julia Blas – Instituto de Desarrollo y Diseño INGAR (UTN-CONICET). E-mail: mariajuliablas@santafe-conicet.gov.ar.

Federico Hauque – Universidad Tecnológica Nacional, Facultad Regional Santa Fe. E-mail: fghauque@gmail.com.

Santiago Re – Universidad Tecnológica Nacional, Facultad Regional Santa Fe. E-mail: lsantire2@gmail.com.

Marta Castellaro. Universidad Tecnológica Nacional, Facultad Regional Santa Fe. E-mail: mcastell@frsf.utn.edu.ar.

978-1-5386-3057-0/17/$31.00 © 2017 IEEE

En este contexto, como parte del trabajo práctico integrador del año 2016, la cátedra de AEDD solicitó a los alumnos la implementación de un juego (denominado King Coder) basado en la detección de errores de sintaxis sobre códigos escritos en C++. Tomando como base los trabajos entregados por los diferentes grupos de alumnos, los docentes de la cátedra decidieron (durante el ciclo lectivo 2017) extender la propuesta precedente a fin de desarrollar una herramienta de evaluación que sea de utilidad para los objetivos de la cátedra. La extensión de la herramienta se diseñó de forma tal que se pueda implementar con los contenidos desarrollados en la asignatura. De esta manera se posibilita contar con un instrumento de evaluación y seguimiento, construido en base a lo desarrollado por los mismos alumnos y, además, se habilita a utilizar el desarrollo como ejemplo para que los alumnos puedan ver una aplicación de los contenidos dictados. En este trabajo se presenta el diseño de la herramienta propuesta junto con la implementación de un primer prototipo que ayuda a llevar a cabo el proceso de evaluación continua de los contenidos dictados en la materia. De esta manera, la herramienta final permitirá suplantar los cuestionarios implementados vía formularios web, posibilitando la corrección automática de diferentes tipos de ejercicios.

El resto del trabajo se encuentra estructurado de la siguiente manera. En la sección 2 se presenta una breve descripción de los errores de programación más frecuentes a fin de contextualizar el objeto de estudio de la herramienta propuesta. La sección 3 describe los contenidos desarrollados en la cátedra poniendo énfasis en la forma de evaluación y en la herramienta propuesta. La sección 4 presenta el juego que impulsó el desarrollo de la herramienta, el cual fue implementado por los alumnos del año 2016 en el marco del trabajo práctico integrador. La sección 5 introduce el diseño propuesto para la herramienta final, poniendo especial énfasis en las funcionalidades a implementar y en el proceso de desarrollo a seguir. La sección 6 muestra el primer prototipo desarrollado en base a los objetivos propuestos. Finalmente, la sección 7 sintetiza los trabajos futuros y las conclusiones.

II. Errores de Programación

El ideal en programación es producir programas legibles, libres de errores, que funcionen en forma correcta y puedan ser modificados con un mínimo de pruebas [15]. Existen cuatro momentos durante el proceso de desarrollo en los cuales pueden detectarse errores, a saber:

1) Antes que el programa sea compilado.
2) Mientras el programa se compila.
3) Mientras el programa se ejecuta.
4) Después que el programa ha sido ejecutado y se ha examinado la salida.

Una forma de detectar errores antes de que un programa sea compilado es mediante el uso de la verificación de escritorio. La verificación de escritorio (o también llamada prueba de escritorio) refiere al proceso de verificar el código fuente en busca de errores inmediatamente después que este ha sido especificado. Por su parte, los errores detectados por el compilador se conocen como errores en tiempo de compilación o errores de sintaxis, puesto que corresponden a violaciones de sintaxis (es decir, de reglas gramaticales) propias del lenguaje de programación elegido (por ejemplo, omitir un punto y coma es un error de sintaxis en el lenguaje C++) [16]. Los errores asociados a la programación que se dan mientras el programa se ejecuta se conocen de manera formal como errores en tiempo de ejecución o errores de lógica. Tales errores refieren a los problemas que se evidencian cuando el programa se ha diseñado sin prever la resolución de fallas. Finalmente, los errores que se observan cuando se analizan los resultados obtenidos (luego de que el programa ha sido ejecutado), corresponden a una comprensión incorrecta de los requerimientos de los usuarios. Esta incomprensión da como consecuencia que la lógica utilizada para su codificación no sea la correcta y, por lo tanto, los resultados obtenidos de la ejecución no se correspondan con los resultados esperados.

III. Programación en C++: Aspectos Claves a Trabajar en 1º Año

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup [17]. Se basa en el lenguaje de programación C e incluye una serie de mecanismos que permiten al programador la manipulación de objetos. Por este motivo, desde el punto de vista de los lenguajes orientados a objetos, C++ se clasifica como híbrido.

Las características propias de C++ lo convierten en uno de los primeros lenguajes de programación que aprenden los desarrolladores. Como parte de la formación de grado, la cátedra de AEDD propone utilizar este lenguaje para introducir a los alumnos en la programación de sistemas de información simples. Tal como se ha enunciado con anterioridad, la propuesta consiste en la codificación de problemas sencillos (bajo el formato de ejercicios prácticos) junto con la implementación de un aplicativo grupal desarrollado de forma incremental a lo largo de múltiples etapas (bajo el formato de trabajo práctico integrador). En este proceso de aprendizaje, se busca no sólo que los alumnos adquieran las destrezas básicas que todo programador debe tener, sino que además interioricen conceptos tales como estructuras de control, tipos de datos abstractos y diseño de estructuras de datos dinámicas. La Tabla 1 detalla los contenidos dictados en el cursado (anual) de la materia.

El desarrollo de los contenidos planificados involucra múltiples instancias de evaluación, las cuales se desarrollan bajo la forma de exámenes parciales y cuestionarios. En ambos casos, es usual solicitar que el alumno trabaje sobre un conjunto de códigos previamente definidos a fin de evaluar tanto su nivel de entendimiento como su capacidad de análisis. Sin embargo, frecuentemente los alumnos presentan dificultades al analizar códigos desarrollados por otras personas. Por este motivo, durante el ciclo lectivo 2016, la cátedra impulsó a sus alumnos a desarrollar un juego llamado King Coder en virtud de tener una herramienta de soporte para la resolución de este tipo de problemas construida por los propios alumnos.

IV. King Coder: Un Juego Educativo como parte del Trabajo Práctico Integrador

El trabajo práctico integrador del año 2016 se formuló con la idea de que los alumnos desarrollen una herramienta lúdica que habilite la competencia entre diferentes grupos de usuarios. Los juegos incluidos como parte de dicha herramienta posibilitarían a los usuarios el desarrollo de habilidades tales como aprender mecanografía, agilizar el manejo de operadores lógicos, relacionales y aritméticos, y analizar código fuente en busca de errores. Este último caso corresponde a King Coder. Un sistema de puntos fue ideado a fin de premiar las partidas de cada uno de los juegos.

De acuerdo con la consigna planteada, los jugadores de King Coder debían revisar y corregir líneas de código de un algoritmo codificado en C++ elegido al azar de un conjunto de algoritmos disponibles. El objetivo fue reforzar la lógica del jugador y su capacidad para entender/interpretar código ajeno.

Al iniciar el juego, la herramienta muestra al jugador un código escrito en C++ (cuyas líneas se encuentran numeradas comenzando en 1) que incluye errores de sintaxis. El jugador debe indicar el número de las líneas en las cuales considera que existen errores. Es importante destacar que el juego no indica la cantidad de errores presentes, por lo que el usuario debe analizar detalladamente el contenido del código a fin de encontrar todos los errores existentes. Una partida queda definida por cinco códigos distintos elegidos de forma aleatoria. Si las respuestas son todas correctas y, además, no se omitió ninguna línea con error, la partida termina en forma exitosa y el usuario gana puntos. En caso contrario, el usuario no obtiene ningún puntaje. Al finalizar la partida, el juego muestra una estadística que indica cuantos aciertos tuvo el usuario en cada uno de los códigos evaluados junto con el puntaje obtenido.

Como parte de la consigna, la cátedra brindó un archivo de texto con 100 códigos C++ recopilados de las entregas que realizaron los alumnos en etapas precedentes. Este archivo debía ser usado como información de entrada durante la implementación del juego a fin de obtener los códigos a ser analizados. Además, se especificó que la implementación del juego debía realizarse como una librería de la herramienta final. De esta forma, los desarrollos resultantes serían fácilmente adaptables a nuevos contextos.

V. Diseño de la Herramienta

Tal como se mencionó con anterioridad, tomando como base los desarrollos realizados por los alumnos en el trabajo práctico integrador del año 2016, en 2017 el equipo de docentes y auxiliares puso en marcha el desarrollo de una herramienta de software para brindar soporte al proceso de evaluación continua previsto por la cátedra. El objetivo de esta herramienta es agilizar la construcción, realización y análisis de resultados de evaluaciones temáticas, en las cuales los alumnos sean capaces de resolver un conjunto de ejercicios previamente especificados (ya sea por los docentes bajo el formato de cuestionario de evaluación o por los propios estudiantes para practicar sus habilidades bajo el formato de juego). Así, la aplicación propuesta tendrá dos formas de ser utilizada: como herramienta/mecanismo de evaluación y como juego de aprendizaje. De acuerdo con esta conceptualización, la herramienta final permitirá que:

1) Los estudiantes:
   - *a* - Adquieran una mayor destreza al leer códigos ajenos, entrenando sus habilidades de comprensión e interpretación de forma lúdica.
   - *b* - Tengan a mano un mecanismo de evaluación que les resulte familiar y que les permita afrontar los cuestionarios de la cátedra de forma clara y concisa, sin generar conflictos de interpretación sobre el conjunto de consignas propuestas.
   - *c* - Puedan resolver cuestionarios y/o evaluaciones usando una herramienta de software desarrollada, en parte, por sus compañeros de años precedentes.

2) Los docentes:
   - *a* - Cuenten con una base de ejercicios clasificados de acuerdo a su tipo, complejidad y tema que pueda ser utilizada como instrumento de enseñanza o evaluación para otros fines.
   - *b* - Puedan construir cuestionarios de evaluación de forma dinámica, utilizando un mecanismo de configuración basado en los ejercicios disponibles.
   - *c* - Posean un mecanismo de análisis de resultados automatizado que les permita conocer indicadores claves del desempeño de los alumnos en relación a los contenidos evaluados.

La existencia de una base de ejercicios disponibles ahorra tiempo al momento de construir el cuestionario de evaluación, haciendo que el docente no deba preocuparse por redactar nuevos códigos ni diagramar nuevos tipos de ejercicios. El software elegirá aleatoriamente un subconjunto de problemas del conjunto de los ejercicios disponibles haciendo uso del tipo, complejidad y tema especificado por el docente al momento de generar la evaluación. Sin embargo, también permitirá que el docente seleccione un subconjunto específico de ejercicios, en el caso que así lo desee, para ser incluidos en la evaluación. La base de ejercicios podrá ser ampliada en cualquier momento, flexibilizando la cantidad de problemas disponibles al momento de realizar esta selección. Se trabajará sobre tres tipos de problemas, a saber:

1) Búsqueda de errores de sintaxis.
2) Ejercicios verdadero/falso.
3) Problemas de opciones múltiples.

De acuerdo con esta clasificación, se tendrá un conjunto de ejercicios asociado a cada uno de los temas detallados en la Tabla 1. Los docentes configurarán evaluaciones haciendo uso de un conjunto de parámetros predefinidos. Los alumnos utilizarán estas configuraciones al realizar una evaluación y generarán un reporte de la misma. Luego, los docentes podrán ver los resultados obtenidos de forma global o particular, según lo requieran.

La Fig. 1 visualiza las funcionalidades planificadas como parte de la herramienta de software propuesta en relación a cada uno de los usuarios identificados.

A. Estrategia de Desarrollo

El desarrollo basado en prototipos permite al cliente evaluar en forma temprana el producto de software (bajo construcción) e interactuar con diseñadores y desarrolladores para determinar si el desarrollo actual cumple (o no) con sus expectativas [19]. Utilizando esta estrategia se obtienen dos beneficios. Por un lado, los desarrolladores tienen la posibilidad de trabajar con el cliente su entendimiento sobre las funcionalidades de la herramienta a construir lo que habilita un desarrollo mucho más medido y consensuado del aplicativo final. Por otra parte, el cliente tendrá resultados visibles en el corto plazo. Esto facilita su entendimiento sobre el funcionamiento de la herramienta. Al incorporar desarrollo incremental sobre el modelo evolutivo se puede controlar la complejidad y los riesgos. En este caso, se desarrolla una parte del producto en una etapa y se reserva el resto para el desarrollo futuro. De esta manera, los prototipos que se construyen son mecanismos de prueba de un conjunto de funcionalidades que condensan la idea principal del sistema en desarrollo, aumentando su funcionalidad con el paso del tiempo. Esta forma de trabajo posibilita construir aplicaciones de software de forma gradual las cuales, si son diseñadas correctamente, pueden dar lugar a diferentes versiones del aplicativo en distintos momentos del proceso de desarrollo.

El esquema de diseño propuesto en la Fig. 1 identifica un conjunto de funcionalidades que pueden ser encapsuladas en diferentes módulos. Esto permite diagramar una estrategia de desarrollo basada en prototipos en la cual los diferentes prototipos incorporen, de forma incremental, un conjunto de módulos creciente a medida que se avance con el desarrollo. Dado que los módulos no son completamente independientes, es importante definir (en una primera instancia) las estructuras de datos y los mecanismos de almacenamiento que darán soporte a la totalidad de la herramienta diseñada. Además, para el caso de módulos interdependientes, debe planificarse cuidadosamente el cronograma de desarrollo y posterior despliegue a fin de evitar inconvenientes e incompatibilidades.

Son múltiples las ventajas de utilizar este tipo de estrategia de desarrollo en el contexto de la herramienta propuesta. La construcción de prototipos no sólo permitirá evaluar la adecuación de las soluciones parciales (habilitando la posibilidad de extender a los alumnos versiones tempranas del software para facilitar su incorporación como parte de las herramientas de la cátedra), sino que además posibilitará la incorporación de nuevas funcionalidades a medida que se avance en el desarrollo de los mismos. También permitirá que los alumnos prueben el aplicativo a fin de realizar un proceso de validación y verificación sobre cada uno de los módulos implementados. De esta forma, se logrará un testeo de los prototipos que agilizará el proceso de desarrollo y permitirá avanzar sabiendo que no se han presentado inconvenientes mayores. Complementariamente, los alumnos que se involucren en esta actividad podrán aportar nuevas funcionalidades a ser incorporadas en el diseño de la herramienta. Estas funciones podrán ser codificadas por los mismos estudiantes, planificándose su incorporación como parte de algún prototipo, lo que genera un espacio didáctico para motivar a los alumnos en el proceso de aprendizaje.

B. Especificación de Funcionalidades

1) Usuario #1: Docente

A fin de acceder al conjunto de funcionalidades de administración, este tipo de usuario deberá pasar previamente un proceso de autenticación.

TABLA I
CONTENIDOS CURRICULARES DE AEDD

Cuatrimestre	*Semana*	*Presentación*
1	1	- Presentación de la materia. - Introducción.
	2	- Resolución de problemas generales. - Algoritmos. - Expresiones lógicas.
	3	- PSeInt[a]. - Estructuras de control condicionales.
	4	- Estructuras de control anidadas. - Problemas de algoritmia.
	5	- Paradigmas. - Lenguajes de programación.
	6	- Proceso de programación. - Lenguaje C++.
	7	- Datos de entrada/salida. - Tipos de datos simples. - Estructuras de control condicionales.
	8	- Estructuras de control repetitivas C++.
	9	- Problemas de secuencias.
	10	- Descomposición modular. - Funciones.
	11	- Funciones. - Generación de números aleatorios.
	12	- Paso de parámetros por copia.
	13	- Paso de parámetros por referencia. - Recursividad.
	14	- Caracteres. - Cadenas de caracteres.
	15	- Arreglos.
2	16/17	- Operaciones sobre arreglos.
	18	- Estructuras y registros.
	19	- Ordenamiento.
	20	- Complejidad.
	21-23	- Matrices.
	24	- Tipo de datos abstractos.
	25	- Punteros.
	26/27	- Archivos.
	28	- Listas basadas en arreglos. - Pilas basadas en arreglos. - Colas basadas en arreglos.
	29	- Estructuras de datos dinámicas.
	30	- Listas basadas en estructuras de datos dinámicas. - Pilas basadas en estructuras de datos dinámicas. - Colas basadas en estructuras de datos dinámicas.
	31	- Problemas combinados.

[a] PSeInt es una herramienta de pseudocódigo que ayuda a los estudiantes a centrar su atención en los conceptos fundamentales de la algoritmia computacional [18].

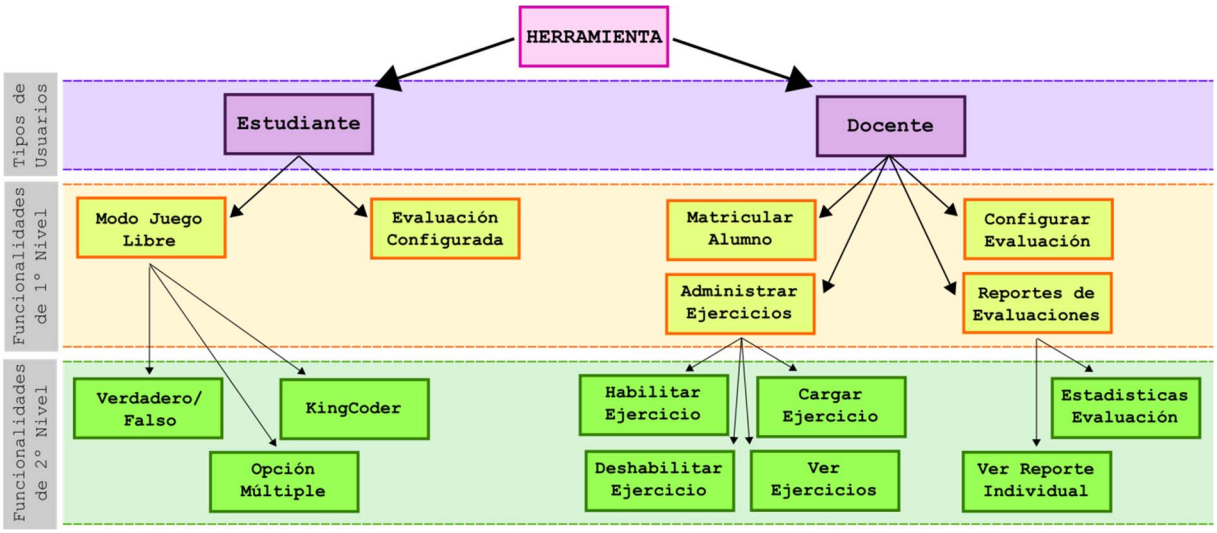

Fig. 1. Esquema de la herramienta de software propuesta en este trabajo que destaca las principales funcionalidades planificadas para cada tipo de usuario identificado junto con el nivel de accesibilidad de las mismas.

1) *Matricular Alumno*: Teniendo en cuenta que los alumnos deberán realizar cuestionarios de evaluación haciendo uso de la herramienta propuesta, es necesario tener un mecanismo que posibilite su identificación como generadores de reportes. Aún más, es deseable contar con la información de los alumnos a fin de poder, posteriormente, medir su evolución en términos del desempeño obtenido en múltiples reportes. Por este motivo, se propone que los docentes generen un listado de los alumnos habilitados para realizar evaluaciones configuradas en la herramienta. De esta manera, sólo podrán hacer uso de esta opción aquellos estudiantes que pertenezcan al listado. Es importante destacar que esta funcionalidad no implica la creación de perfiles de usuarios, sino que simplemente le dará al docente la posibilidad de matricular alumnos en la herramienta a fin de habilitarles la posibilidad de desarrollar evaluaciones.
2) *Administrar Ejercicios*: Cada uno de los ejercicios incluidos en la base de problemas de la herramienta será identificado por una categoría, la cual quedará definida de acuerdo a su tipo, complejidad y tema. Dentro de cada categoría se incorporará un identificador secuencial a fin de facilitar el proceso de selección aleatoria.
   a - *Cargar Ejercicio*: El docente podrá incorporar nuevos ejercicios a la base de problemas existentes por medio de la utilización de esta funcionalidad. Para esto, además del enunciado del ejercicio, deberá indicar el tipo de problema, su complejidad y el tema al cual está asociado. El identificador dentro de la categoría se generará automáticamente al finalizar el proceso de carga, haciendo uso de la cantidad de ejercicios existentes. Este identificador será informado al docente. Posteriormente, se actualizará dicha cantidad a fin de indicar que un nuevo ejercicio ha sido introducido a la base de problemas.
   b - *Habilitar / Deshabilitar Ejercicio*: Los ejercicios incorporados en la base de problemas podrán tomar uno de dos estados, a saber: i) Habilitado (es decir, podrá ser elegido como parte de una evaluación –ya sea por medio de un proceso de selección aleatorio o de forma manual-). ii) Deshabilitado (en ningún caso podrá ser elegido como parte de una evaluación).
   Para modificar el estado de los ejercicios, el docente deberá utilizar esta funcionalidad. Esto permitirá que aquellos ejercicios que ya han sido utilizados en evaluaciones pasadas, no puedan ser seleccionados para evaluaciones futuras. En diferentes momentos, un mismo ejercicio podrá tener diferentes estados.
   c - *Ver Ejercicio*: El docente podrá ver los ejercicios a fin de experimentar como se visualizan los problemas como parte de una evaluación. Para hacer uso de esta funcionalidad deberá indicar, dentro de la categoría de ejercicios deseada, el identificador del problema a visualizar.
3) *Configurar Evaluación*: Para que los alumnos puedan realizar una evaluación, el docente debe generar una configuración que indique el tenor de la misma. Para esto, la configuración deberá especificar:

   1. Para cada categoría de ejercicios disponibles (es decir, para cada categoría en la que existan ejercicios habilitados): cantidad de ejercicios y formato de selección (manual / automático).
   2. Utilizar temporizador (si/no).
   3. Cantidad de respuestas correctas para aprobar.
   4. Visualización de respuestas correctas (si/no).
   5. Visualización de resultados (si/no).
   6. Permitir reorganizar ejercicios (si/no).

   Si se configura el formato de selección manual, el docente deberá especificar el conjunto de identificadores de los ejercicios a ser incluidos en la evaluación. Al habilitar el uso del temporizador, el estudiante tendrá una cantidad de segundos (especificada en respuesta a dicha configuración) para resolver los problemas. Por otra parte, si se permite reorganizar ejercicios, el alumno podrá saltar una única vez cada uno de los problemas presentados en

la evaluación. Los ejercicios omitidos deberán responderse obligatoriamente al final del cuestionario en el mismo orden en el que fueron saltados. En caso que no esté habilitada la reorganización de ejercicios, los problemas deberán ser resueltos en el orden especificado (por el docente o por el proceso de selección aleatorio).

4) *Reportes de Evaluaciones*: Se especifican dos funcionalidades asociadas.
   - *a* - *Estadísticas Evaluación:* En base a los reportes generados como resultado de las evaluaciones realizadas por los alumnos, los docentes tendrán la posibilidad de generar diferentes tipos de planillas con distintos objetivos, a saber: i) Reporte de aprobados: Se visualiza en una tabla el estado (aprobado/no aprobado) de cada alumno junto con el porcentaje de respuestas que han sido correctas. ii) Estadísticas generales: Haciendo uso de todos los reportes, se calcula un conjunto de indicadores de interés a nivel grupal a fin de obtener información sobre el desempeño general de los alumnos.
     Entre las estadísticas propuestas se incluyen:
     1. Cantidad de ejercicios de un tema específico resueltos correctamente por tipo y complejidad.
     2. Si la evaluación se generó con un conjunto específico de ejercicios, cantidad de respuestas correctas para cada ejercicio.
     3. Cantidad de ejercicios de una complejidad dada que no han sido resueltos correctamente.
     4. Tema en el que se obtuvo mayor cantidad de respuestas incorrectas.
     5. Cantidad total de ejercicios incorrectos.
   - *b* - *Ver Reporte Individual*: El docente podrá visualizar, para un alumno en particular, el reporte de resultados obtenidos en una evaluación específica. Para indicar el reporte a visualizar, el docente utilizará el identificador de usuario. Una vez encontrado el reporte, la aplicación mostrará cada uno de los ejercicios planteados durante la evaluación junto con el estado obtenido (resolución correcta/resolución incorrecta) y, en el caso de haber sido resuelto incorrectamente, se visualizará la opción elegida por el alumno. Al finalizar, se mostrará en formato de tabla un resumen de la evaluación.
     Haciendo uso de esta funcionalidad en conjunto con un estudio de resultados, el docente podrá analizar la evolución del desempeño del alumno a lo largo de múltiples instancias de evaluación.

2) Usuario #2: Estudiante

Este tipo de usuario no requerirá ningún tipo de autenticación para utilizar la herramienta. Los estudiantes podrán acceder libremente al modo juego, mientras que para acceder a las evaluaciones configuradas deberán pertenecer al conjunto de estudiantes matriculados por los docentes.

1) *Modo Juego Libre*: Incluye cuatro juegos, a saber:
   - *a* - *King Coder*: Para cumplir con este módulo se utilizará el desarrollo realizado por los alumnos en el ciclo 2016 como parte del trabajo práctico. Las interfaces lúdicas construidas por los estudiantes junto con el sistema de recompensas y puntajes implementado servirán como soporte del juego.
   - *b* - *Verdadero / Falso*: Se implementará un nuevo juego cuya modalidad consistirá en la resolución de ejercicios del tipo verdadero / falso. El sistema de puntajes asignados aún no se encuentra especificado. El desarrollo de este módulo se propone como parte del trabajo práctico del ciclo lectivo 2017.
   - *c* - *Opción Múltiple*: Se implementará un nuevo juego en el que los usuarios deberán seleccionar una opción (de un conjunto de alternativas) en respuesta a un ejercicio. En este contexto, durante el ciclo lectivo 2015 los alumnos desarrollaron una aplicación similar a Preguntados [21] que permitía al usuario responder preguntas vinculadas a contenidos académicos (específicamente las cátedras de primer año de Ingeniería en Sistemas de Información). Este nuevo juego se denominó PregUTNados [22] e impulsó el desarrollo de juegos como parte de los trabajos prácticos de cátedra. Tomando como base estos aplicativos, se propone adaptar tales implementaciones a fin de utilizarlas como parte del módulo propuesto en la nueva herramienta.
2) *Evaluación Configurada*: Haciendo uso de la configuración interna especificada por los docentes, el alumno deberá resolver un conjunto de ejercicios predeterminados. No podrá dejar problemas sin responder, pudiendo alterar el orden de los ejercicios según lo detallado en la configuración realizada.
   Al finalizar la evaluación se le solicitará al usuario que indique su código de matriculación. Como resultado, la herramienta generará un reporte de resultados cuyo nombre estará constituido por el código indicado.

C. Requerimientos Adicionales

Como complemento de los requerimientos planteados para las funcionalidades de la Fig. 1, se detalla una serie de requerimientos adicionales en vistas de garantizar el funcionamiento deseado de la herramienta propuesta.

La encriptación de todos los archivos que se generen como mecanismo de almacenamiento de información será clave para impedir accesos no deseados. Desde los reportes de evaluación hasta los ejercicios y valores de configuración, se utilizarán esquemas de encriptación para garantizar su seguridad e impedir a que los alumnos alteren o visualicen su contenido.

Por otra parte, la accesibilidad de la herramienta será un aspecto clave a trabajar, considerando que en los años precedentes la cátedra ya ha tenido que generar evaluaciones para alumnos no videntes (año 2016). En este contexto, se propone simplificar las interfaces de la herramienta de forma tal que los lectores de pantalla comúnmente utilizados (como es el caso del lector JAWS [20]) puedan traducir su contenido sin inconvenientes. Esto posibilitará que la herramienta pueda ser utilizada por personas no videntes.

VI. Prototipo de la Herramienta (Versión 1)

A. Estructuras de Datos

Para gestionar los datos a manipular en la herramienta es vital organizar y modelar la información en diferentes estructuras. Como se ha indicado, con el objetivo que esta herramienta sea también un recurso didáctico para que los alumnos revisen y comprendan, las estructuras y modelos de datos a utilizar serán seleccionados e implementados haciendo uso de los contenidos de la materia.

En primer lugar, una de las entidades más relevantes de la herramienta bajo desarrollo son los códigos asociados a las consignas. La Fig. 2 visualiza el esquema de la estructura propuesta. Sin importar la modalidad de juego a la que correspondan, todos los códigos deberán tener un identificador, el texto del código propiamente dicho, un estado (habilitado o deshabilitado), un nivel de dificultad y el tema de la cátedra al cual está asociado. Para facilitar su manipulación interna también será necesario almacenar la cantidad de líneas definidas como parte del código. Además, dado que existen tres tipos de problemas diferentes, es necesario distinguir la categoría a la cual pertenece cada código. Para esto se utilizó un valor entero (denominado "tipo") que discrimina los ejercicios de opción múltiple, de los de verdadero/falso y los de King Coder. En el caso que la instancia de la estructura represente un ejercicio de opción múltiple, un vector almacena el texto de las opciones de la pregunta. En caso contrario, dicho vector permanecerá vacío. Por último, se definió una cadena de caracteres que almacena información útil a fin de identificar la respuesta correcta del problema. Para el caso de King Coder dicha cadena guardará una secuencia de valores que representan los números de línea donde se hallan errores. Para los juegos restantes, la cadena llevará un único valor que será el número de opción correcta o un carácter V o F.

Código	
Tipo de dato	**Nombre**
vector <string>	texto
int	tipo
vector<string>	opciones
string	correcta
int	cantLineas
string	tema
int	dificultad
string	id
int	estado

Fig 2. Estructura de datos que representa un código fuente.

Alumno	
Tipo de dato	**Nombre**
int	id
string	nombre
string	apellido
char	comision
int	año

Fig. 3. Estructura de datos que indica un alumno matriculado.

Otra entidad importante a tener en cuenta son los alumnos matriculados en la herramienta, ya que su identificación es requisito indispensable para determinar el acceso a una evaluación. Los datos básicos como su nombre, apellido, comisión y año, acompañados de su número de identificación, constituyen la información a básica a ser almacenada para reconocer un estudiante. La estructura de datos propuesta para guardar esta información se esquematiza en la Fig. 3.

Por otra parte, es necesario representar los ejercicios que forman parte de una evaluación. La estructura que modela esta información se visualiza en la Figura 4 y sirve para que los docentes puedan configurar las distintas evaluaciones en términos del tema y la dificultad. Los códigos que aparecen en las evaluaciones pueden elegirse de dos formas. La primera consiste en determinar el tipo de juego (es decir, el tipo de ejercicio) junto con el tema y rango de dificultad de las consignas (definidos conjuntamente como un par de valores). El programa elegirá aleatoriamente los códigos de una evaluación limitando su búsqueda a los que cumplen con las características dadas. La segunda forma de definir un ejercicio es indicar exactamente el conjunto de identificadores de los códigos a evaluar. Esto se define en el vector denominado "especificas". Bajo esta configuración, no se ejecuta un proceso de selección aleatorio y el docente es capaz de elegir el conjunto de ejercicios que desea evaluar.

Un conjunto de ejercicios define una evaluación. Cuando un docente configura una evaluación, es importante que toda la información que la describe quede almacenada en una estructura independiente (Fig. 5). Tal estructura debe guardar un identificador, los ejercicios a resolver, el estado general, su descripción, un vector con las soluciones de los alumnos que respondieron la evaluación y el tiempo máximo (en segundos) disponible para responder (si se encuentra configurado).

Por último, la estructura que define la solución de una evaluación (esto es, el reporte) se describe en la Fig. 6. Esta estructura se encuentra vinculada a un alumno específico (es decir, el alumno que resolvió la evaluación) por medio de su identificador. Además, almacena el vector que indica la resolución del conjunto de ejercicios propuestos y la fecha en la que el alumno resolvió la evaluación.

B. Almacenamiento

Las estructuras de datos definidas en la sección precedente deben guardarse en archivos persistentes para su uso en ejecuciones posteriores de la herramienta. Se decidió usar un archivo por cada tipo de estructura definida, con la excepción del almacenamiento de ejercicios. Estos últimos serán guardados junto con las evaluaciones de acuerdo al siguiente criterio: para cada evaluación se guardará primero su definición y, luego, todos los ejercicios que contiene.

En todos los casos el contenido de los archivos será encriptado para garantizar la seguridad de acceso a la información. Aunque la misma estructura podría ser almacenada en una única base de datos SQL, a fin de lograr una herramienta didáctica que pueda ser integrada con desarrollos de alumnos de primer año, se eligió trabajar con el enfoque basado en archivos.

Ejercicio	
Tipo de dato	**Nombre**
vector<int>	tipo
vector<string>	tema
pair<int,int>	dificultad
int	cantidad
vector<string>	especificas

Fig. 4. Estructura de datos que representa un ejercicio que forma parte de una evaluación.

Evaluación	
Tipo de dato	**Nombre**
vector<Ejercicio>	contenido
int	cantidadTotal
int	id
bool	estado
string	comentario
vector<Resolucion>	resoluciones
int	tiempo_preg

Fig. 5. Estructura de datos que representa una evaluación.

Resolucion	
Tipo de dato	**Nombre**
int o string	id_alumno
int o string	id_evaluacion
vector<int>	respuestas
date	fecha

Fig. 6. Estructura de datos que representa la resolución de una evaluación por parte de un alumno.

C. Implementación

1) Grupo de Trabajo

El diseño e implementación de la herramienta se realizó inicialmente por los docentes y ayudantes de cátedra que trabajaron en el desarrollo del trabajo práctico del año 2016.

En principio, se planteó el diseño general de la herramienta y, posteriormente, los ayudantes organizaron las estructuras básicas y desarrollaron el primer prototipo a fin de generar una primera aproximación del funcionamiento requerido. El módulo correspondiente a King Coder fue el primero en ser incorporado. Para esto se tomó como base el desarrollo de uno de los grupos del año 2016 y, sobre esta codificación, se realizaron las adaptaciones requeridas para dar lugar a las funcionalidades restantes de la modalidad "Juego Libre".

2) Lenguaje y Entorno de Desarrollo (C++/ZinjaI)

Teniendo en cuenta que la cátedra de AEDD utiliza el lenguaje C++ para enseñar a programar sus alumnos y, en virtud de construir una herramienta basada en parte de los desarrollos de sus estudiantes, la herramienta propuesta será desarrollada haciendo uso del mismo lenguaje. Además se utilizará el IDE ZinjaI como herramienta de desarrollo, el cual corresponde al entorno utilizado durante las clases prácticas de la materia. Esto facilitará la integración, a futuro, de nuevos módulos y/o funcionalidades.

3) Diseño del Proyecto

Para simplificar la implementación de la herramienta propuesta, es necesario separar las funcionalidades a programar en distintos módulos. De esta forma se logra separar la ejecución de funciones estrechamente relacionadas en porciones de código que pueden trabajar con cierto nivel de independencia. Así entonces, se simplifica su codificación.

La Fig. 7 esquematiza los archivos propuestos para implementar los módulos antes mencionados como librerías C++ junto con sus relaciones. A excepción del código principal, los archivos restantes indicados con extensión ".cpp" tienen asociados archivos de cabecera (es decir, archivos con extensión ".h") que ofrecen una interfaz para usar las funciones de la librería cuando una ejecución lo necesite. Es importante destacar que las estructuras Código, Evaluación y Ejercicio se implementaron en forma de tipo de dato abstracto a fin de compartir sus comportamientos con mayor facilidad. La Tabla 2 resume las responsabilidades asociadas a cada librería propuesta.

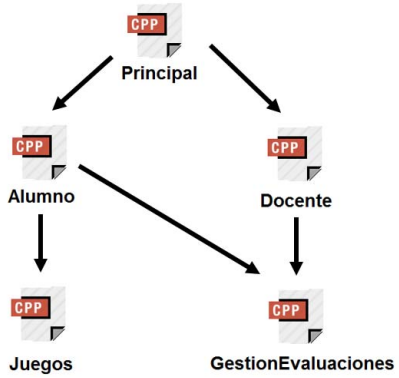

Fig. 7: Esquema de archivos propuestos para las librerías.

El archivo de código Principal.cpp contendrá la función que indicará el flujo de ejecución principal de la herramienta (es decir, la función "main()"). Además incluirá los distintos procedimientos que faciliten la navegación por el menú principal y la autenticación como usuario alumno matriculado por el docente. Las interfaces visuales se implementarán por separado en una librería visual, la cual contendrá las funciones que permitirán dibujar en pantalla la interfaz del programa. Tal librería será incluida en todos los códigos que requieran interactuar con el usuario/jugador.

Si el usuario decide identificarse como docente, la ejecución pasará del módulo Principal.cpp a la librería Docente.cpp. Esta librería brinda las funciones de matricular/desmatricular alumnos, así como también brinda soporte a la carga, modificación y eliminación de ejercicios. Si el docente desea crear una nueva evaluación, se ejecutarán funciones de la librería GestionEvaluaciones.cpp. Tales funciones permiten configurar las preferencias de los ejercicios a plantear en la prueba, como así también la modificación y eliminación de evaluaciones existentes. Además calcula datos estadísticos relevantes sobre las evaluaciones resueltas (es decir, sobre los reportes), devolviendo los resultados detallados a la librería Docentes.cpp para la confección de tablas que indiquen el rendimiento global/particular de los alumnos.

Por el contrario, si el usuario se identifica como alumno, la ejecución de instrucciones se da en la librería Alumno.cpp. Esta librería es capaz de visualizar y dar inicio a una evaluación gestionada desde el módulo GestionEvaluaciones.cpp. Para esto, la librería imprime en pantalla uno a uno los ejercicios configurados, pasando su resolución a la librería Juegos.cpp. Tal librería se encuentra encargada de verificar las respuestas de los usuarios según el tipo de juego involucrado, devolviendo a la librería Alumno.cpp la correctitud de la respuesta. Una vez finalizada la evaluación, el módulo Alunmo.cpp se vale del módulo GestionEvaluaciones.cpp para completar los datos de las respuestas del alumno en un reporte (esto es, una instancia de la estructura "Resolución").

TABLA II
PRINCIPALES FUNCIONALIDADES ASOCIADAS A LAS LIBRERÍAS PROPUESTAS

Archivo de código	*Funcionalidades*
Principal	- Arranque de la ejecución. - Menú principal. - Autenticación de usuarios.
Docente	- Gestión de ejercicios. - Gestión de alumnos.
Gestión Evaluaciones	- Creación de evaluaciones. - Cálculo de estadísticas. - Elaboración de reportes. - Gestión de soluciones.
Alumno	- Listado de evaluaciones disponibles. - Coordinación de la ejecución de evaluaciones.
Juegos	- Determinación de la correctitud de las respuestas.

4) Desarrollo de la Versión 1

Tal como se enunció con anterioridad, la primera versión de la herramienta incluye el King Coder y las primeras versiones de los juegos verdadero/falso y opción múltiple.

Aunque las reglas para jugar son simples y los problemas pueden parecer sencillos, el nivel de detalle requerido en el análisis de cada código convierte a cada uno de los juegos en un desafío adecuado a las posibilidades cognitivas de los estudiantes; no siendo ni tan simple como para que lo desechen, ni tan complejo como para desanimarlos.

Al elegir el módulo King Coder el programa imprime en pantalla un código C++ simple y corto con todas sus líneas numeradas. Seguidamente pide al usuario que indique los números de líneas en las cuales considera que existen errores de sintaxis (Fig. 8). No hay un número fijo de líneas con errores. Lo único que sabe el usuario es que existe al menos un error en todo el código. En base a los valores indicados, el programa determina la correctitud o no de la solución.

En el caso de ejecutar la opción 2, el programa imprime la consigna de un problema de programación junto con una propuesta de solución. Ante este ejercicio el usuario debe analizar cuidadosamente el código y determinar la veracidad de la consigna dada (Fig. 9). Aun siendo muy simple en su planteo, este juego no es tan fácil como parece. Un detalle no observado puede invalidar todo el código, dando como consecuencia una respuesta incorrecta. Tal como se ha mencionado con anterioridad, un desarrollo más avanzado de este módulo se dará en el marco del trabajo práctico de 2017.

Finalmente, en el juego de opciones múltiples (Code Quiz) la herramienta imprime un código C++ acompañado de una pregunta y 4 posibles respuestas. El usuario debe seleccionar una única opción dentro de las respuestas propuestas. (Fig. 10). Al igual que en los casos precedentes, la resolución de este tipo de ejercicios no es una tarea trivial. De hecho, la dificultad es mucho mayor que en los casos anteriores. La pregunta que se debe responder puede referirse a cualquier característica propia del código que se imprima en pantalla, ya sea su complejidad, eficiencia, cantidad de ejecuciones de bucles, y demás particularidades propias de la codificación del programa. Esta característica trae como consecuencia un mayor dinamismo al momento de plantear interrogantes.

Fig. 8. Problema planteado en el módulo King Coder.

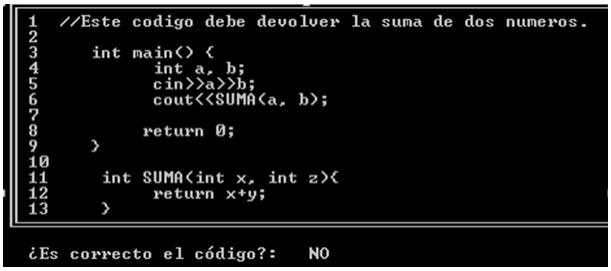

Fig. 9. Problema planteado en el módulo verdadero/falso.

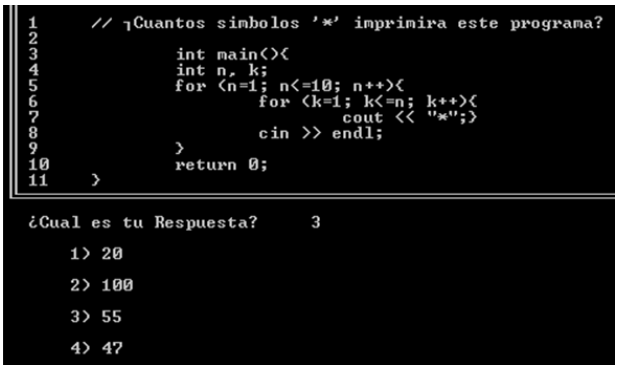

Fig. 10. Problema planteado en el módulo de opción múltiple.

D. Pruebas

Actualmente se está llevando a cabo el proceso de pruebas asociado a cada uno de los módulos implementados. Una vez finalizado este proceso, se pedirá a los alumnos de AEDD 2017 que realicen una prueba del prototipo actual e indiquen sus comentarios u observaciones en una breve encuesta.

VII. Conclusiones y Trabajos Futuros

Se ha presentado el diseño de una herramienta de soporte a la enseñanza de programación basada en desarrollos propios de la cátedra AEDD. La herramienta ha sido pensada como complemento al proceso de evaluación continua requerido en los primeros años de la carrera, dando lugar a un mecanismo alternativo de evaluación formulado en base a problemas detectados durante el dictado de los contenidos. A futuro se espera completar la implementación con el desarrollo de las funcionalidades faltantes. Es importante destacar que el prototipo actual cumple con los requerimientos básicos establecidos para un proceso de evaluación inicial. Sus funcionalidades actúan como cimientos constructivos de la herramienta final, posibilitando el desarrollo de las funcionalidades faltantes sobre la implementación actual sin requerir mayores modificaciones.

El desarrollo basado en prototipos evolutivos permitirá realizar pruebas en diferentes etapas de desarrollo sobre diferentes módulos. Estas pruebas podrán ser internas (es decir, entre los docentes, ayudantes y becarios de grado asociados a la cátedra) o podrán darse como experiencias frente al curso para que los alumnos realicen una retroalimentación de la versión de la herramienta que actualmente se encuentra bajo desarrollo. De esta forma, será posible mejorar los diferentes módulos a medida que se avance con las distintas etapas de implementación. El diseño propuesto posibilita la incorporación de los juegos pendientes (ejercicios verdadero/falso y problemas de opciones múltiples) como nuevos módulos a ser incluidos dentro del prototipo actual. Como parte del desarrollo colaborativo planificado, la cátedra propondrá a los alumnos del ciclo 2017, la mejora del módulo verdadero/falso como parte del trabajo práctico. Complementariamente se adaptarán más funcionalidades de los desarrollos del año 2015 para mejorar el módulo de opción múltiple. Ambas implementaciones serán, a futuro, incorporadas a la herramienta final siguiendo un esquema de trabajo similar al realizado con el módulo King Coder.

La herramienta final se pondrá a disposición de quienes quieran utilizarla, ya sean docentes de otras facultades regionales como así también alumnos que deseen evaluar sus conocimientos haciendo uso de los juegos propuestos. Se redactará un manual de usuario que indique el alcance de cada configuración posible, a fin de garantizar que los docentes de otros establecimientos puedan configurar correctamente la herramienta de acuerdo a sus necesidades particulares.

Aunque la versión actual de la herramienta incorpora un conjunto reducido de funcionalidades, se espera probar su efectividad al momento de evaluar conocimientos en la primera evaluación de la cátedra. Además, se considera que la existencia del aplicativo contribuirá a mostrarles a los alumnos un desarrollo producido por sus propios compañeros y con las mismas herramientas que ellos utilizan, lo que se cree que contribuirá a su motivación para la participación en el desarrollo de nuevos proyectos. Este trabajo presenta la propuesta a fin de difundirla en la comunidad educativa no sólo para divulgar la herramienta sino también para enriquecer los aspectos trabajados con nuevas perspectivas.

Referencias

[1] T. Jenkins, “The Motivation of Students of Programming”, Proc. Innovation and technology in computer science education (ITiCSE’ 01), pp. 53-56,2001.

[2] J. Beltrán, H. Sánchez, and M. Rico, “Análisis cuantitativo y cualitativo del aprendizaje de Programación I en la Universidad Central del Ecuador”, Revista Tecnológica-ESPOL, vol.28, no.5, pp. 194-210, 2015.

[3] A. Gomes and A. J. Mendes, “An environment to improve programming education”, Proc. International conference on Computer systems and technologies (CompSysTech '07), p. 88, 2007.

[4] L. E. Margulieux, R. Catrambone, and M. Guzdial, “Employing subgoals in computer programming education”, Computer Science Education, vol. 26, no. 1, pp. 44-67, 2016.

[5] W. Sohn, “Design and Evaluation of Computer Programming Education Strategy using Arduino”. Advanced Science and Technology Letters, vol. 66, no. 1, pp. 73-77, 2014.

[6] A. Cypher and D. C. Halbert, Watch what I Do: Programming by Demonstration, MIT Press, 1993.

[7] R. G. McDaniel and B. A. Myers, “Getting More out of Programming-by-demonstration”, Proc. Conference on Human Factors in Computing Systems (SIGCHI ‘99), pp. 442–449, 1999.

[8] M. B. Rosson and C. D. Seals, “Teachers As Simulation Programmers: Minimalist Learning and Reuse”, Proc. Conference on Human Factors in Computing Systems (SIGCHI ‘01), pp. 237–244, 2001.

[9] H. Lieberman, Your Wish is My Command: Programming by Example, Morgan Kaufmann, 2001.

[10] M. Travers, “Recursive Interfaces for Reactive Objects”, Proc. Conference on Human Factors in Computing Systems (SIGCHI ‘94), pp. 379–385, 1994.

[11] J. Montemayor, “Physical Programming: Software You Can Touch”. Proc. Conference on Human Factors in Computing Systems (SIGCHI ‘01), pp. 81–82, 2001.

[12] Campus UTN-FRSF, https://campusvirtual.frsf.utn.edu.ar/. [Fecha de último acceso 17/04/2017].

[13] Uri Online Judge, https://www.urionlinejudge.com.br. [Fecha de último acceso 17/04/2017].

[14] ZinjaI, http://zinjai.sourceforge.net/. [Fecha de último acceso 17/04/2017].

[15] G. J. Bronson, C++ ingeniería y ciencias, International Thomson, 2000.

[16] W. A. Savitch, Resolución de problemas con C++, Pearson, 2007.

[17] B. Stroustrup, The C++ Programming Language, 1997.

[18] Pseint, http://pseint.sourceforge.net/. [Fecha de último acceso 17/04/2017].

[19] R. Pressman, Software Engineering: A Practitioner´s Approach, 7th ed. McGraw-Hill, 2010.

[20] JAWS,http://www.freedomscientific.com/Products/Blindness/JAWS. [Fecha de último acceso 21/06/2016].

[21] Preguntados, http://www.triviacrack.com. [Fecha de último acceso 17/04/2017].

[22] F. Hauque, and G. Saudejaud, “Una herramienta de soporte al aprendizaje”, Proc. Congreso Nacional de Ingeniería Informática / Sistemas de Información (CONAIISI ’16), 2016.