30

AUTOMATIZACIÓN DE LOS PROCESOS DE CORRECCIÓN Y AUTOEVALUACIÓN DE PRÁCTICAS EN ASIGNATURAS CON CONTENIDOS DE PROGRAMACIÓN MEDIANTE HERRAMIENTAS TIC

AUTOMATION OF CORRECTION PROCESSES AND SELF-EVALUATION OF ASSIGNMENTS IN SUBJECTS WITH PROGRAMMING CONTENT THROUGH ICT TOOLS

Christopher Expósito-Izquierdo
cexposit@ull.edu.es
Israel López-Plata
ilopezpl@ull.edu.es
Belén Melián-Batista
mbmelian@ull.edu.com
José Marcos Moreno-Vega
jmmoreno@ull.edu.com

Universidad de La Laguna, España

RESUMEN

La programación informática se ha convertido en los últimos años en una herramienta transversal en múltiples áreas de conocimiento. A través de la programación el estudiante crea programas que presentan un comportamiento deseado en un contexto práctico concreto. Sin embargo, el proceso de escritura de programación le demanda al estudiante habitualmente de conocimientos de múltiples áreas, dominio de lenguajes de programación, algoritmos de propósito específico y lógica formal, entre otros.
Actualmente, la Universidad de La Laguna incluye la programación en múltiples titulaciones oficiales. En estas titulaciones el estudiante se enfrenta a la realización de múltiples prácticas de laboratorio donde tiene que demostrar sus conocimientos relativos a programación para la resolución de algún problema planteado por el profesorado dentro del contexto de la asignatura que imparte.
El proceso de corrección de las prácticas involucra un análisis exhaustivo por parte del profesorado de las propuestas de programación realizadas por el alumnado. Esto habitualmente implica la corrección de un número elevado de propuestas durante las sesiones prácticas, lo cual da lugar a que el nivel de detalle en la evaluación sea inevitablemente inferior al deseado. Consecuentemente, se da lugar a que algunos alumnos presenten insatisfacción con las calificaciones obtenidas así como extenuación por parte del profesorado ante la carga de trabajo concentrada en las sesiones prácticas.
En este trabajo se plantea el diseño, implementación y validación de una herramienta software que automatiza los procesos de corrección a la vez que facilita la autoevaluación por parte del alumnado durante el desarrollo de las prácticas.

PALABRAS CLAVE: test de unidad; código fuente; software; algoritmo; proceso de evaluación; feedback.

ABSTRACT

In recent years, computer programming has become a transversal tool in multiple areas of knowledge. The student creates programs that present a desired behavior in a given practical context through the programming. However, the writing process demands the student usually of knowledge about multiple areas, domaining programming languages, algorithms of specific purpose, and formal logic, among others.
Nowadays, the Universidad de La Laguna includes computer programming in multiple official degrees. In these degrees the student faces the performance of multiple laboratory practices where he has to demonstrate his knowledge related to computer programming for solving a problem proposed by the teacher within the context of the subject he teaches.
The process of correcting the practices involves a thorough analysis by the teachers of the programming proposals made by the students. This usually involves assessing a large number of proposals during the practice sessions, which results in the level of detail in the evaluation being inevitably lower than desired. Consequently, some students are dissatisfied with the grades obtained as well as exhaustion by the teachers in the face of the workload concentrated in the practical sessions.
In this paper, the design, implementation, and validation of a software tool that automates the assessment processes while facilitating self-assessment by students during the development of practices is described.

KEYWORDS: unit test; source code; software; algorithm; assessment process; feedback.

INRODUCCIÓN

El proceso de enseñanza-aprendizaje ha evolucionado de forma notable durante las últimas décadas en el contexto universitario. En particular, se ha ido progresivamente abandonando el denominado *modelo tradicional* (Valero y Navarro, 2010). El alumnado dentro de este modelo era un elemento totalmente pasivo. A su vez el profesor era el guía elemental del proceso de enseñanza. Su labor se centraba en enseñar los contenidos establecidos y llevar a cabo el proceso de evaluación. Este proceso se caracterizaba por una escasa o nula retroalimentación hasta el final del mismo y donde las horas de clase presencial impartidas regían el número de créditos necesarios para la obtención de las titulaciones. La evaluación en este proceso de enseñanza-aprendizaje era consecuentemente de carácter sumativo.

El tradicional modelo de enseñanza-aprendizaje universitario ha sido reemplazado progresivamente por un modelo educativo en donde el profesorado aplica nuevas estrategias de enseñanza así como nuevas aproximaciones destinadas a la evaluación de las competencias requeridas. Al mismo tiempo, la contabilización del trabajo realizado se ha desplazado también desde el número de horas presenciales impartidas por el profesorado al trabajo demandado al alumnado para adquirir las competencias establecidas. Todo ello ha dado lugar a que hoy en día el alumnado se haya convertido en un elemento activo y, por tanto, también protagonista de su aprendizaje. Por otro lado, el profesorado ha pasado a ser un elemento guía, evaluador y facilitador de retroalimentación.

Uno de los aspectos que mayor cambio ha sufrido a raíz del abandono de los procesos tradicionales de enseñanza-aprendizaje es la evaluación. La educación superior ha experimentado e incorporado durante los últimos años un gran número de nuevas y disruptivas formas de evaluación del aprendizaje. Caben ser destacadas en este contexto la coevaluación, la evaluación entre pares o la autoevaluación, entre otras. Sin embargo, en muchas ocasiones las universidades se enfrentan a grandes desafíos para la incorporación de este tipo de aproximaciones. Algunos de estos desafíos vienen marcados por establecimientos normativos destinados a la especificación pormenorizadas de todos los aspectos relativos al proceso de evaluación.

La retroalimentación juega un papel central dentro de los nuevos procedimientos de evaluación. Ésta permite al alumno comprobar a lo largo del tiempo cómo evoluciona y cuál es el grado de logro de las competencias requeridas para la obtención de una titulación. Al mismo tiempo, el alumno puede comprobar cuáles son los objetivos alcanzados y cuáles restan por ser alcanzados dentro de su aprendizaje. La demanda cada vez mayor de retroalimentación por parte del alumnado hace que el profesorado deba idear nuevas vías para facilitar en una medida oportuna la totalidad del proceso de aprendizaje. A su vez, tal como indica (Gómez et al., 2013), el profesorado debe determinar aquellos mecanismos más adecuados para la concienciación del alumnado en

materia de identificación de objetivos de aprendizaje y de su proceso. En concreto, la identificación de aquellos aspectos a potenciar y de aquellos puntos débiles en el proceso de aprendizaje se hace vital para acometer la adquisición de competencias correspondiente. Todo ello proporciona múltiples beneficios para el proceso. Por un lado, el profesorado en todo momento puede comprobar cómo y en qué grado su alumnado adquiere las competencias demandadas. Por otro lado, el alumnado recibe una retroalimentación regular de su actividad (Noche et al., 2017).

En los últimos años han aparecido diversas propuestas por parte de la comunidad educativa acerca de cómo proporcionar retroalimentación al alumnado. Sin embargo, la rúbrica destaca por ser una herramienta de evaluación con gran potencial y versatilidad (Gordillo y Rodríguez, 2006). A grandes rasgos, en una rúbrica se recogen las puntuaciones que obtienen los alumnos en base a su rendimiento o resultados alcanzados. El empleo de estas guías de puntuación permite determinar claramente las expectativas sobre el desempeño académico del alumnado, cómo progresa y, en última instancia, proporcionar retroalimentación. Al mismo tiempo, el empleo de rúbricas en el proceso de evaluación facilita el mismo, lo hace más transparente, garantiza la equidad y reduce la ambigüedad (Jácome, 2013). El conjunto de todas estas ventajas ha hecho que las rúbricas sean consideradas como un elemento atractivo para la evaluación en la educación superior, tanto por parte del profesorado como del alumnado.

Una vez realizado un breve recorrido por los diferentes modelos de enseñanza-aprendizaje así como por la introducción de la relevancia de la retroalimentación en materia de evaluación, en el resto de este trabajo se va a describir y analizar cómo puede automatizarse la corrección de prácticas de programación informática en el contexto de la educación superior y en particular dentro de la Universidad de La Laguna (Ala-Mutka, 2005). Con este objetivo en mente, el resto de este trabajo se organiza tal como sigue. La Sección 2 introduce la programación informática y su relevancia en un amplio abanico de titulaciones universitarias. La Sección 3 presenta la motivación y objetivos que dan lugar a la incorporación de herramientas de automatización de prácticas de programación informática. Posteriormente, se analiza una aproximación basada en tests unitarios para la evaluación de los resultados de prácticas de programación en la Sección 4. Luego, se describe una herramienta software para automatizar la evaluación en la Sección 5. Finalmente, la Sección 6 presenta algunas conclusiones extraídas de la realización de este trabajo así como varias líneas de trabajo futuro.

PROGRAMACIÓN INFORMÁTICA

El aumento de la productividad en la realización de la práctica totalidad de tareas de nuestro día a día ha sido una constante a lo largo de la evolución humana. La irrupción de la tecnología ha permitido que tareas realizadas tra-

dicionalmente a mano pasen a ser realizadas mediante máquinas con el fin de mejorar la calidad de vida de las personas. Los sistemas de computación (ordenadores, teléfonos inteligentes, tabletas, etc.) han dado lugar en este contexto a que las tareas de carácter intelectual hayan ido siendo también reemplazadas de forma paulatina, y no únicamente aquellas asociadas al trabajo físico.

Una gran parte de los cambios más destacados en los modelos productivos durante los últimos años han venido de la mano de la introducción y desarrollo de sistemas de computación. Éstas son herramientas tecnológicas que, apoyadas en elementos físicos, están destinadas para la ejecución de programas informáticos, procesando grandes cantidades de información de forma ágil y precisa. En este sentido, un programa informático es una secuencia de instrucciones que el sistema de computación es capaz de comprender y que conforma los pasos que debe realizar el sistema para realizar una tarea específica (Sudkamp, 1988). Entre estas tareas caben ser destacadas por su relevancia el almacenamiento, organización, procesamiento y transmisión de información.

Tradicionalmente los programas informáticos han sido desarrollados por programadores; profesionales encargados de diseñar, probar, implementar, mantener y mejorar los programas informáticos en un contexto dado. Desde un punto de vista formal, la programación informática hace referencia al proceso de diseño, codificación y mantenimiento de programas informáticos que permiten resolver un problema planteado (Wirth, 1976). La escritura de programas informáticos requiere el uso de lenguajes de programación (Robins et al., 2003). Éstos son lenguajes cuyos elementos básicos y reglas de asociación entre los mismos están formalmente definidos, evitándose por ello la aparición de cualquier tipo de ambigüedad. A continuación se puede comprobar en la Figura 1 un ejemplo básico de un código de programación que, tras ser compilado, muestra el mensaje «Hola a todos» cuando es ejecutado.

HelloWorld.java ×

Source | History

```
1  package es.ull.programming;
2
3  /**************************************************************************
4   * Compilación: javac HelloWorld.java Ejecución: java HelloWorld
5   *
6   * Muestra el mensaje "Hola a todos"
7   *
8   * @author Christopher Expósito-Izquierdo (christopherexpositoizquierdo@gmail.com)
9   *
10  **************************************************************************/
11 public class HelloWorld {
12
13     public static void main(String[] args) {
14         System.out.println("Hola a todos");
15     }
16 }
17
```

Figura 1. Ejemplo de código de programación.

Durante los últimos años se ha producido un proceso de democratización de la programación informática. Esto es, la programación informática

ha traspasado los ámbitos meramente tecnológicos y científicos en el que ha estado habitualmente confinada. Este proceso ha venido motivado por sus notables beneficios para el desarrollo personal e intelectual (Lau & Yuen, 2009). En particular, la programación informática tiene claros beneficios desde el punto de vista pedagógico. En primer lugar, el desarrollo de competencias asociadas a la programación permite estructurar el pensamiento (Lye & Koh, 2014). Esto posibilita la descomposición de problemas complejos en otros más sencillos y de fácil resolución. Al mismo tiempo, estimula la creatividad y refuerza el pensamiento crítico, así como la autonomía en el aprendizaje. La programación también mejora la capacidad de atención y la concentración, así como fomenta el orden y la capacidad de comprensión, a la vez que confiere mayor autonomía e interés por la experimentación y el aprendizaje. Finalmente, la programación informática también potencia la capacidad de cálculo y lógica. Por otro lado, las competencias digitales abren un amplio abanico de posibilidades laborales y económicas al alumno derivadas de los cambios en el modelo productivo y la transformación digital.

La programación informática ha sido durante las últimas décadas una herramienta imprescindible para la sistematización de tareas, para la comunicación, así como para la adecuada gestión de información. En particular, la programación informática ha permitido que un sistema de computación ejecute tareas anteriormente realizadas de forma manual y con un consecuente alto coste económico y temporal. A su vez, la programación informática es una herramienta que posibilita la exploración y extracción de conocimiento entre grandes volúmenes de datos.

La programación informática se ha convertido en los últimos años en una herramienta transversal en múltiples áreas de conocimiento. A través de la programación el estudiante crea programas que presentan un comportamiento deseado en un contexto práctico concreto. Sin embargo, el proceso de escritura de códigos de programación le demanda al alumno habitualmente conocimientos altamente transversales. Entre éstos cabrían ser destacados el dominio de lenguajes de programación, aritmética, algoritmos de propósito específico y lógica formal, entre otros. Apoyando lo dicho, vivimos en la era de la tecnología y podemos observar que los conocimientos en la informática son los más demandados a día de hoy, propiciando de esta manera que sea necesario que estos nuevos conocimientos y utilidades se hayan empezado a impartir en los últimos años como materias universitarias en titulaciones como Matemáticas o Ingeniería electrónica entre otras.

En la actualidad, la programación es una materia transversal a múltiples áreas de conocimiento y consecuentemente está recogida en los diversos planes de estudios de múltiples titulaciones universitarias. Éste es el caso del Grado en Ingeniería Informática, Grado en Ingeniería Electrónica o el Grado en Matemáticas, entre otros. En dichas titulaciones el alumnado se enfrenta a múltiples materias en las que tienen que demostrar sus conocimientos de programación mediante la presentación de supuestos

prácticos en forma de aplicaciones informáticas de limitado alcance que deben funcionar con corrección para que sean consideradas como válidas.

MOTIVACIÓN Y OBJETIVOS

El aprendizaje de la programación informática es un tópico que ha ganado un alto interés durante los últimos años dentro de diversos ámbitos. En particular, la programación informática es una disciplina técnica de relevancia dentro del conjunto de competencias de cualquier ingeniería actual, en especial en la ingeniería informática. Este interés es tal que algunos autores como Malik (Malik, 2000), afirman incluso que la enseñanza de la programación informática y de lenguajes de programación no está y nunca estará obsoleta. Al mismo tiempo, otros autores como Maheshwari (Maheshwari, 1997) indican que el aprendizaje cualitativo de lenguajes de programación tiene un rol central en la mejora de la calidad de los productos software y del desarrollo de la profesión de la ingeniería del software. A pesar del enorme interés por el aprendizaje de la programación informática, ésta se enfrenta a un gran desafío: es una habilidad. Por tanto, el aprendizaje en sí mismo requiere una gran inversión temporal y además no es la única motivación que desarrollan los alumnos de cursos de programación informática.

Adquirir las competencias asociadas a la programación informática plantea diversos retos que han sido estudiados en la literatura durante los últimos años. Por ejemplo, Jakovljevic (Jakovljevic, 2003) señala que los métodos actuales destinados al aprendizaje de la programación informática no son en general efectivos. También, Humphrey (Humphrey, 1997) concluye que los alumnos no adquieren las destrezas suficientes en materia de programación informática porque el proceso de enseñanza-aprendizaje fomenta que el alumno aprenda un nuevo lenguaje de programación sin dedicar un tiempo adecuado a la escritura de programas que hagan que se consoliden los conceptos. Por último, Howell (Howell, 2003) analiza el impacto que la aproximación educativa propuesta y la configuración de contextos de aprendizaje para la enseñanza de un lenguaje de programación tiene en la adquisición de competencias.

A pesar de las propuestas planteadas en la literatura destinadas a la mejora de la adquisición de competencias en materia de programación informática, actualmente se adolece de herramientas tecnológicas que sean verdaderas facilitadoras del proceso de enseñanza-aprendizaje. En este contexto, la incorporación de herramientas de corrección automática es un campo de estudio prometedor por sus numerosas ventajas. Sin embargo, para identificar adecuadamente el potencial de estas herramientas hay que analizar primeramente el proceso de corrección de códigos de programación en el marco de la docencia de programación informática.

En la actualidad la Universidad de La Laguna incluye la programación informática en múltiples titulaciones oficiales, así como cursos de extensión. Éste es el caso del Grado en Ingeniería Informática, Grado en Matemáticas o el Grado en Ingeniería Electrónica. En estas titulaciones el alumnado se enfrenta a la realización de múltiples prácticas de laboratorio donde tiene que demostrar sus conocimientos relativos a programación informática para la resolución de algún problema planteado por el profesorado dentro del contexto de la materia que imparte.

El proceso de corrección de las prácticas involucra un análisis complejo y exhaustivo por parte del profesorado de las propuestas de programación informática realizadas por el alumnado. Esto habitualmente implica la corrección de un número elevado de propuestas durante las sesiones de prácticas, lo cual da lugar a que el nivel de detalle en la evaluación sea inevitablemente inferior al deseado. Consecuentemente, se da lugar a que algunos alumnos presenten insatisfacción con las calificaciones obtenidas, así como extenuación por parte del profesorado ante la carga de trabajo concentrada en las sesiones prácticas. Algunos de estos comportamientos son analizados en profundidad en (Lance et al., 1990), (Dennis et al., 1996) y (Brown et al., 2004), entre otros. Específicamente, en estos trabajos se describen aquellos contextos en que el evaluador humano se encuentra influenciado por la primera impresión causada por la respuesta del alumno, el tipo de relación previa que evaluador y alumno han podido tener y la ortodoxia de las respuestas proporcionadas, respectivamente.

Desafortunadamente, la limitación temporal y de recursos humanos existente en las titulaciones de educación superior hace que aspectos como la corrección minuciosa o la inspección del buen uso de prácticas de programación, utilización de estructuras o algoritmos requeridos, entre otros, no sea siempre posible o se lleve a cabo tan solo de forma parcial. En este contexto, la incorporación de herramientas de las tecnologías de la información y las comunicaciones (TIC) destinadas a la corrección de prácticas por parte del profesorado y la autoevaluación del alumnado hace que se obtenga un proceso de enseñanza-aprendizaje más equitativo y eficaz que mediante los actuales métodos de corrección. Esto se debe a que las herramientas TIC para la evaluación automática de prácticas de programación informática aportan un marco objetivo y carente de errores y prejuicios al proceso de evaluación del alumnado.

Con el objetivo de mejorar el proceso de corrección de prácticas de programación informática, en este trabajo se plantea el desarrollo e incorporación de una herramienta TIC que automatice los procesos de corrección, sea elemento de asistencia durante los mismos, a la vez que facilite la autoevaluación por parte del alumnado durante el desarrollo de las asignaciones prácticas. De forma más específica, el objetivo concreto perseguido con la realización de este trabajo es proporcionar una herramienta TIC que (i) reduzca el tiempo de corrección de las prácticas de programación, (ii) aumente el nivel de precisión en la evaluación y (iii)

constituya un marco de autoevaluación para el alumnado. Cabe señalar que, a pesar de ser aplicada en el contexto de la Universidad de La Laguna, la herramienta creada puede ser aplicada de forma transparente en otros ámbitos.

Finalmente, es importante destacar que la metodología llevada a cabo durante el desarrollo del presente trabajo consiste en una primera fase de planificación donde se ha evaluado el contexto de aplicación del mismo y las necesidades derivadas del proceso actual de enseñanza-aprendizaje. La fase de desarrollo permite el diseño e implementación de la herramienta TIC para la corrección de las prácticas de programación. En este sentido, la herramienta desarrollada es proporcionada al alumnado antes de la realización de las prácticas junto con la documentación relativa a la misma para que puedan familiarizarse con su uso. Finalmente, durante el proceso de evaluación de las diferentes prácticas a desarrollar se empleará la herramienta creada para cumplir con los diferentes objetivos planteados en el trabajo.

TEST DE UNIDAD

Satisfacer los objetivos de evolución de un proyecto de programación informática en tiempo y forma es un gran desafío al que se enfrentan los alumnos durante las sesiones prácticas. Por ello, la incorporación de herramientas TIC que agilicen este proceso es de especial relevancia. Éstas favorecen la adquisición de las competencias requeridas y ayudan a mitigar el alto nivel de abandono asociado a este tipo de formación (Keengwe et al., 2008). En este caso, la herramienta de automatización de la corrección y autoevaluación desarrollada en este trabajo está basada en el uso de *tests de unidad* (Runeson, 2006). Durante los últimos años, la aparición de las conocidas como *metodologías ágiles* –principios establecidos para alcanzar los requisitos del software a través de procesos incrementales e interactivos– ha favorecido el uso de los tests de unidad durante las diferentes fases del desarrollo de software. Algunas de las metodologías ágiles que han favorecido el uso de los tests de unidad son la Programación extrema (Beck, 2000), Scrum (Rising & Janoff, 2000) y Kanban (Ahmad et al., 2013), entre otras.

Los tests de unidad son elementos destinados a la verificación objetiva de la corrección de determinadas secciones de un código de programación. Habitualmente, los tests de unidad son desarrollados por el mismo programador que implementa el código fuente a evaluar. Una vez el programador ha creado el código de su software, éste diseña un conjunto de tests que, en primera instancia, persiguen garantizar que todas las secciones del código desarrollado funcionan de la manera requerida y, además, que su integración da como resultado el comportamiento global esperado. El resultado por tanto de la ejecución de un test de unidad so-

bre un código de programación desarrollado por un alumno proporciona como salida la superación o el fallo del test en cuestión. En particular, la no superación de un test de unidad indica que el código de programación bajo análisis no satisface los requisitos funcionales impuestos por dicho test. Consecuentemente, se puede afirmar que el código fuente analizado es incorrecto. De manera similar, un código de programación supera un test de unidad sólo si éste satisface los requisitos impuestos en el test. Sin embargo, la superación de un test de unidad tiene un resultado con un ámbito únicamente local. Esto es, tan solo indica que el código de programación presenta el comportamiento esperado en el escenario recogido en el test. Incorrecciones no contempladas en un test de unidad no pueden ser en forma alguna descartadas a través de su mera superación. Para evitar este tipo de inconsistencias, se debe desarrollar una batería suficientemente amplia y heterogénea de tests de unidad que permita abarcar todas las secciones del código de programación. Por último, cabe señalar que la aplicación de los tests de unidad para un código de programación se realiza durante una fase de testeo.

Los tests de unidad pueden perseguir diferentes objetivos de evaluación al ser aplicados sobre un código de programación. Entre estos objetivos caben ser destacados la comprobación de métodos de acceso, el análisis de la correcta implementación de algoritmos, la evaluación de métodos de entrada y salida de datos, el análisis de la eficiencia de implementación, o la usabilidad de interfaces gráficas, entre otras. Sin embargo, en el caso de la herramienta desarrollada en este trabajo se ha optado por proponer tests de unidad con el objetivo de llevar a cabo un análisis de resultados esperados. En concreto, los tests de unidad que integran la herramienta propuesta persiguen comparar el resultado obtenido por los algoritmos desarrollados por los alumnos durante las sesiones prácticas con aquellos establecidos por el profesor de la materia. De esta manera, la superación o no de los tests propuestos por el profesor evidencia la corrección en los desarrollos realizados por los alumnos.

El análisis de resultados esperados ha sido elegido como el objetivo de los tests de unidad debido a la metodología aplicada habitualmente en la mayor parte de las sesiones prácticas de materias relacionadas con la programación informática en instituciones universitarias y, en concreto, en la Universidad de La Laguna. En particular, se requiere a los alumnos el desarrollo de códigos de programación con funcionalidades modestas durante los primeros cursos de las titulaciones. Algunos ejemplos pueden ser la realización de operaciones de cálculo, lectura y escritura de datos en ficheros de texto, creación de pequeñas aplicaciones visuales o la creación de conversaciones automáticas. La complejidad de los objetivos propuestos al alumno va aumentando con el paso de las sesiones prácticas y el desarrollo de las materias. De esta manera, el profesor suele requerir en fases últimas el desarrollo de soluciones software que integren componentes

simples anteriormente desarrollados por el alumno. Esto ha demostrado ser una herramienta de motivación e interés por la materia. Sin embargo, también hace que el alumno por lo general centre su atención en el correcto comportamiento de sus desarrollos y abandone aspectos esenciales como la comprobación de parámetros, la gestión de errores y la organización y documentación del código, entre otros. Esto también se debe a que las pruebas realizadas a los códigos de programación implementados por los alumnos suelen ser escasas y limitadas a secciones principales. La consecuencia de esta situación es que los objetivos establecidos por el profesor para las sesiones prácticas no se satisfagan en su totalidad. Como se describe en la sección previa, el elevado número de alumnos y las limitaciones humanas y temporales impide que se evalúen todos los detalles de los códigos sometidos a evaluación.

HERRAMIENTA DE EVALUACIÓN

En esta sección se describe CodeAnalysis, una herramienta software de código abierto que ha sido desarrollada por los autores del presente trabajo con el objetivo de poder automatizar la corrección de prácticas de programación informática así como de brindar la posibilidad de autoevaluación al alumnado.

CodeAnalysis es una herramienta que persigue realizar una evaluación del código fuente programado por un alumno atendiendo a los siguientes criterios:

– Corrección. Evalúa si el programa desarrollado se puede ejecutar y proporciona la salida correcta para los valores de parámetros establecidos por el profesor. En este sentido, el alumno debe ser capaz de realizar su funcionalidad básica con anterioridad a que otros componentes del mismo sean verificados.
– Robustez. Se considera que un programa que implementa un algoritmo es robusto si éste prosigue con la ejecución a pesar de existir valores anómalos en los parámetros de entrada, los sistemas de entrada/salida, etc.
– Complejidad. Hace referencia al total de caminos independientes a través del código fuente del alumno. Un mayor número de puntos de decisión en el código da consecuentemente lugar a una mayor complejidad en el código desarrollado y, a su vez, a un mayor número de puntos de potencial fallo.
– Eficiencia. En términos generales, la demanda de tiempo de computación y de memoria son los dos elementos a destacar en la ejecución de un programa informático. En este contexto, la eficiencia se suele medir mediante el tiempo de procesamiento invertido así como por el espacio de memoria empleado por el programa.

A grandes rasgos, el funcionamiento de CodeAnalysis atiende a los siguientes pasos:

– Las propuestas realizadas por los alumnos de la materia son almacenadas en un directorio de la herramienta durante las sesiones prácticas para proceder a su evaluación automática.
– Se lleva a cabo el proceso de compilación de los diferentes códigos fuentes presentados por los alumnos y almacenados previamente. La compilación de cada uno de los códigos fuentes es realizada de manera automática por CodeAnalysis. Para ello, la herramienta en primer lugar identifica el lenguaje de programación en que está escrito el código fuente y emplea el compilador correspondiente entre aquellos disponibles en el ordenador de ejecución. Cabe señalar que aquellos códigos fuentes que no han podido ser compilados son descartados del proceso de evaluación debido a que no es posible por tanto obtener el programa ejecutable. Habitualmente, como consecuencia, el profesor puede considerar que no se han alcanzado los objetivos más básicos de la práctica planteada y proporcionar una calificación negativa al alumno en cuestión.
– Todas aquellas entregas para los cuales los códigos fuentes correspondientes han superado la fase previa de compilación son sometidas a los tests de unidad disponibles. En concreto, se lleva a cabo la aplicación de cada uno de los tests de unidad que el profesor ha diseñado para la evaluación a todos los programas obtenidos tras la compilación. El resultado de este proceso es la superación o no de cada uno de los tests individuales por parte de los programas sometidos por los alumnos. En este caso, la superación de todos los tests de unidad por parte de un código fuente bajo análisis indicaría que éste se ajusta fielmente a los requerimientos de la sesión práctica. Sin embargo, es habitual que un código fuente proporcionado por un alumno no supere la totalidad de los tests de unidad propuestos. Por tanto, su calificación debe ser proporcional al peso y número de los tests superados.

Los tests de unidad son los elementos centrales de la evaluación de los códigos de programación proporcionados por los alumnos. Sin embargo, para su correcto uso mediante CodeAnalysis, éstos deben poseer un formato conocido. En caso contrario, los tests son descartados durante la aplicación de la herramienta. En particular, un test de unidad válido para ser empleado en CodeAnalysis es un fichero de texto plano que contiene los siguientes campos:

– *name*: nombre del test de unidad.
– *input*: son los valores de los parámetros del código fuente a evaluar. El valor de este campo es dependiente del contexto de la práctica a

evaluar así como de sus objetivos. En algunos casos puede representar una secuencia de números enteros, un fichero de texto plano o una dirección web, entre otros.

– *output*: conforma la salida en formato textual que se espera obtener de la ejecución del código fuente tras llevar a cabo el proceso de compilación y tras haber pasado los valores de parámetros del test de unidad (establecidos en el campo input).

– *dependencies*: define el conjunto de dependencias que tiene el test de unidad con otros tests. En concreto, un test depende de otro si el primero debe ser ejecutado únicamente cuando el segundo ha proporcionado un resultado positivo en la evaluación.

A continuación se muestra en la Figura 2 un ejemplo sencillo de un test de unidad que persigue analizar si un determinado código fuente realiza correctamente la suma de una secuencia de números.

```
#
# Description: It assess the sum of a given sequence of integer numbers
# Author: Christopher Expósito-Izquierdo (cexposit@ull.edu.es)
#

# Name of the test
name = test sum

# Input of the program
input = 7 10 15 -4 8 0 132 -9 -1 15 3

# Expected output
output = 176

# Dependencies
dependencies = test 01, test 02

# End of test
```

Figura 2. Ejemplo de test de unidad destinado a evaluar el resultado de una suma de números enteros.

Como se puede apreciar en este ejemplo, el test creado tiene como nombre 'test sum'. Al mismo tiempo, la secuencia de números enteros a emplear como entrada está definida en el campo input del test. En este caso, la secuencia de números es '7 10 15 -4 8 0 132 -9 -1 15 3'. Por otro lado, el resultado esperado viene definido en el campo output. En este caso, el resultado esperado es 176. Además, es importante resaltar que este test de unidad depende de la correcta aplicación de los tests 'test 01' y 'test 02'. Finalmente, cabe señalar que todas aquellas líneas de texto que se encuentren vacías son descartadas del procesamiento del test de unidad. De forma similar, tal como se puede apreciar en el ejemplo, se pueden incluir comentarios de formato libre en los tests de unidad comenzando las líneas correspondientes con una almohadilla.

Por otro lado, uno de los aspectos destacados de CodeAnalysis es que es capaz de generar automáticamente los programas asociados a los códigos fuentes proporcionados por los alumnos. Con este objetivo en mente, CodeAnalysis hace uso de aquellos compiladores disponibles en el ordenador de ejecución empleado. En concreto, emplea un fichero de texto en formato JavaScript Object Notation (JSON) para describir el conjunto de compiladores a emplear. Este fichero proporciona un conjunto de pares clave-valor que determinan cuál es el compilador a emplear para un lenguaje de programación concreto. De esta manera, CodeAnalysis emplea el compilador correspondiente cuando identifica el lenguaje de programación de un código fuente. Si el lenguaje de programación no se encuentra entre el conjunto de lenguajes definidos en este fichero de compiladores, éste es descartado del proceso de evaluación. Esto ocurre en aquellas situaciones en las que un alumno no ha seguido las indicaciones establecidas por el profesor en cuanto a la elección de lenguaje de programación. A continuación se muestra un ejemplo de fichero de compiladores empleado habitualmente en el proceso de evaluación de prácticas:

```
{
        "java": "javac CODE",
        "c": "gcc CODE -o EXECUTABLE",
}
```


Figura 3. Ejemplo de fichero de compiladores.

En este ejemplo se muestra un fichero de configuración de compiladores empleado por CodeAnalysis en donde se establecen los compiladores para los lenguajes de programación Java y C respectivamente. En cada caso se indica la invocación al compilador correspondiente y dependiente del ordenador de ejecución. Es importante señalar que para cada lenguaje de programación se emplean las etiquetas CODE y EXECUTABLE. Estas etiquetas son empleadas por CodeAnalysis para indicar el lugar que ocupará el nombre del fichero con el código fuente y el nombre del programa ejecutable dentro del proceso de compilación, respectivamente.

En base a lo anteriomente descrito, la evaluación de los códigos fuentes proporcionados por los alumnos requiere de los siguientes elementos:

– Herramienta CodeAnalysis.
– Directorio con los tests a aplicar.
– Fichero en formato JSON con los compiladores existentes en el ordenador de ejecución de la herramienta.
– Directorio con los códigos fuentes realizados por los alumnos a evaluar.

Una vez disponibles los elementos previos se puede llevar a cabo la evaluación de los códigos fuentes incluidos en el directorio correspondiente. Con este objetivo en mente, el usuario debe emplear la interfaz de usuario basada en línea de comandos con la que cuenta CodeAnalysis a través de la consola correspondiente de su sistema operativo. La Figura 4 muestra un ejemplo de ejecución de CodeAnalysis:

```
christopher@christopher-pc:~$ java -jar code-analysis-1.0-SNAPSHOT-jar-with-dependencies.jar -test tests -sc sourcecodes -cps compilers.json -ver true
```

Figura 4. Ejemplo de ejecución de CodeAnalysis para la evaluación de los códigos fuentes ubicados en el directorio sourcecodes.

Como se puede apreciar en la Figura 4, la ejecución de CodeAnalysis requiere emplear el comando *java* acompañado de la opción *-jar* para comenzar la ejecución del fichero JAR de la herramienta. Las restantes opciones permiten establecer el directorio con los tests de unidad empleados en la evaluación, el directorio con los códigos fuentes a evaluar, así como el fichero con la configuración de los compiladores disponibles, respectivamente. CodeAnalysis también proporciona la opción *-ver* acompañado de un valor Booleano para indicar si se desea obtener una salida detallada de la ejecución de CodeAnalysis o tan solo una salida simple de la misma.

La Figura 5 muestra un ejemplo básico de ejecución de la herramienta CodeAnalysis. En este caso, por motivos de simplicidad y espacio se muestra la ejecución de dos tests de unidad para la evaluación de dos códigos fuentes proporcionados por dos alumnos. En particular, los ficheros que contienen los tests de unidad se denominan *tests01.txt* y *tests02.txt*, respectivamente. A su vez, los códigos fuentes de los alumnos están compuestos por un único fichero cada uno con nombres ficticios *andres-lopez-martin.c* y *marta-aguilar-perez.c*, respectvamente.


```
christopher@christopher-pc:~/code-analysis$ java -jar code-analysis-1.0-SNAPSHOT-jar-with-dependencies.jar -test tests -sc sourcecodes -cps ../compilers.json -ver true
Compilers (1):
c : gcc CODE -o EXECUTABLE
Loading tests
        Loading tests from file tests/tests01.txt
        Loading tests from file tests/tests02.txt
        Tests loaded: 1
                /home/christopher/code-analysis/tests/tests01.txt
Analyzing tests
        Tests loaded: 1
                test sum [/home/christopher/code-analysis/tests/tests01.txt]
Loading source codes
        Source codes loaded: 2
                /home/christopher/code-analysis/sourcecodes/marta-aguilar-perez.c []
                /home/christopher/code-analysis/sourcecodes/andres-lopez-martin.c []
Compiling source codes
        /home/christopher/code-analysis/sourcecodes/marta-aguilar-perez.c []
                Command:     gcc /home/christopher/code-analysis/sourcecodes/marta-aguilar-perez.c -o /home/christopher/code-analysis/sourcecodes/executable_marta-aguilar-perez
                Output:
                The souce code has been compiled successfully. The tests can be applied
        /home/christopher/code-analysis/sourcecodes/andres-lopez-martin.c []
                Command:     gcc /home/christopher/code-analysis/sourcecodes/andres-lopez-martin.c -o /home/christopher/code-analysis/sourcecodes/executable_andres-lopez-martin
                Output:
                The souce code has been compiled successfully. The tests can be applied
        Loaded source codes: 2
                /home/christopher/code-analysis/sourcecodes/marta-aguilar-perez.c [/home/christopher/code-analysis/sourcecodes/executable_marta-aguilar-perez]
                /home/christopher/code-analysis/sourcecodes/andres-lopez-martin.c [/home/christopher/code-analysis/sourcecodes/executable_andres-lopez-martin]
Executing tests on source codes
        Source code: /home/christopher/code-analysis/sourcecodes/marta-aguilar-perez.c [/home/christopher/code-analysis/sourcecodes/executable_marta-aguilar-perez]
                Test: /home/christopher/code-analysis/tests/tests01.txt
                        Input (29 characters): |7 10 15 -4 8 0 132 -9 -1 15 3|
                        Expected output (3 characters): |176|
                        Execution command: /home/christopher/code-analysis/sourcecodes/executable_marta-aguilar-perez 7 10 15 -4 8 0 132 -9 -1 15 3
                        Output (3 characters): |208|
                        Result: false
        Source code: /home/christopher/code-analysis/sourcecodes/andres-lopez-martin.c [/home/christopher/code-analysis/sourcecodes/executable_andres-lopez-martin]
                Test: /home/christopher/code-analysis/tests/tests01.txt
                        Input (29 characters): |7 10 15 -4 8 0 132 -9 -1 15 3|
                        Expected output (3 characters): |176|
                        Execution command: /home/christopher/code-analysis/sourcecodes/executable_andres-lopez-martin 7 10 15 -4 8 0 132 -9 -1 15 3
                        Output (3 characters): |176|
                        Result: true
christopher@christopher-pc:~/code-analysis$
```

Figura 5. Resultado de la ejecución de CodeAnalysis en la evaluación de los dos códigos fuentes mediante dos tests de unidad.

La salida de la ejecución mostrada en la Figura 5 se encuentra dividida en las siguientes partes:

– Carga y análisis de tests de unidad. Los tests de unidad disponibles en el directorio especificado a través de la interfaz de línea de comandos son cargados y analizados para comprobar que satisfacen los requerimientos de formato indicados previamente en este trabajo. Aquellos tests de unidad que no cumplan estos requerimientos son descartados del proceso de evaluación. Como se puede observar en el ejemplo de la Figura 5, en este caso, únicamente *tests01.txt* es considerado para llevar a cabo la evaluación de los códigos fuentes.
– Carga y análisis de códigos fuentes. Los códigos fuentes disponibles en el directorio especificado a través de la interfaz de línea de comandos son cargados y analizados para comprobar que se trata de códigos fuentes válidos para su análisis. En particular, aquellos ficheros que no correspondan con códigos fuentes son descartados del proceso de compilación posterior.
– Compilación de códigos fuentes. Todos los archivos de código fuente que superaron la fase previa son compilados empleando el compilador oportuno entre aquellos indicados en el fichero JSON establecido a través de la interfaz de línea de comandos. Para cada uno de los códigos fuentes a compilar se indica explícitamente el comando empleado así como la salida proporcionada por el compilador correspondiente, si la hubiera. Por último, se muestra un listado de todos aquellos programas ejecutables resultantes del proceso de compilación.
– Ejecución de tests de unidad. La última etapa del proceso de ejecución de CodeAnalysis involucra la aplicación de los diferentes tests de unidad a los programas obtenidos de la fase de compilación previa. En este caso, los diferentes tests de unidad válidos son aplicados uno tras otro sobre cada uno de los programas obtenidos. En cada caso, se muestra la entrada pasada al programa en cuestión así como la salida esperada. Al mismo tiempo, se indica el comando de ejecución utilizado y la salida obtenida. Finalmente, se indica si el programa ha superado el test de unidad correspondiente.

La incorporación de esta herramienta en el proceso de evaluación de las prácticas de programación informática hace que la forma de operar se vea modificada en comparación con las prácticas tradicionales. A continuación se describen las principales etapas llevadas a cabo por parte del profesor al emplear CodeAnalysis en su actividad:

– El profesor de la materia diseña un conjunto de tests de unidad suficientemente amplio y destinado a evaluar los diferentes requisitos impuestos a los códigos de programación a crear por los alumnos

durante las sesiones prácticas. La creación de los tests de unidad es una labor que el profesor debe realizar de manera conjunta con la propuesta de práctica.

– Una parte de la batería de tests de unidad creados por el profesor de la práctica son proporcionados al alumnado antes de la evaluación presencial de la misma. Este subconjunto de tests de unidad es conocido como *conjunto de entrenamiento*. Esto permite que los tests de unidad en cuestión sean empleados para la autoevaluación del alumnado durante la fase de desarrollo previa a la evaluación.
– El total del conjunto de tests de unidad es empleado en la sesión presencial para la evaluación de los códigos de programación proporcionados por los alumnos.

En base a lo anteriormente expuesto, es importante resaltar que no se aconseja que el resultado de la aplicación de los tests de unidad se traduzca directamente en la calificación proporcionada al alumno. En su lugar, debe suponer únicamente una parte de su calificación. El resto de la calificación debe atender a la defensa oral, capacidad de acometer cambios imprevistos en el código desarrollado, etc. Estos aspectos deben ser valorados de forma oportuna también por parte del profesor para conformar la calificación última del alumno.

Por otro lado, cabe señalar que CodeAnalysis es una herramienta desarrollada integramente en Java Standard Edition 8. Se trata de un lenguaje de programación sencillo, orientado a objetos, altamente seguro y adecuado para la creación de aplicaciones distribuidas. A su vez, se trata de un lenguaje interpretado por lo que puede ejecutarse de forma homogénea en diferentes plataformas. Esta última característica hace que CodeAnalysis pueda ser empleado por profesores y alumnos que usen habitualmente ordenadores con características de hardware y de software diferentes de una manera similar.

Finalmente, es importante resaltar que con la creación de CodeAnalysis en este trabajo se pretende continuar con la apuesta decidida de la Universidad de La Laguna por la creación y uso de software libre. Con este objetivo en mente, CodeAnalysis es una herramienta distribuida como software libre a través de la plataforma GitHub, mostrada en la Figura 5. Se trata de una plataforma que proporciona servicios de almacenamiento y gestión de versiones de códigos fuentes en entornos colaborativos. Esto hace que se trate de una herramienta especialmente interesante para fomentar la participación de diferentes agentes en el proceso de desarrollo y mejora de las funcionalidades de CodeAnalysis en el futuro.

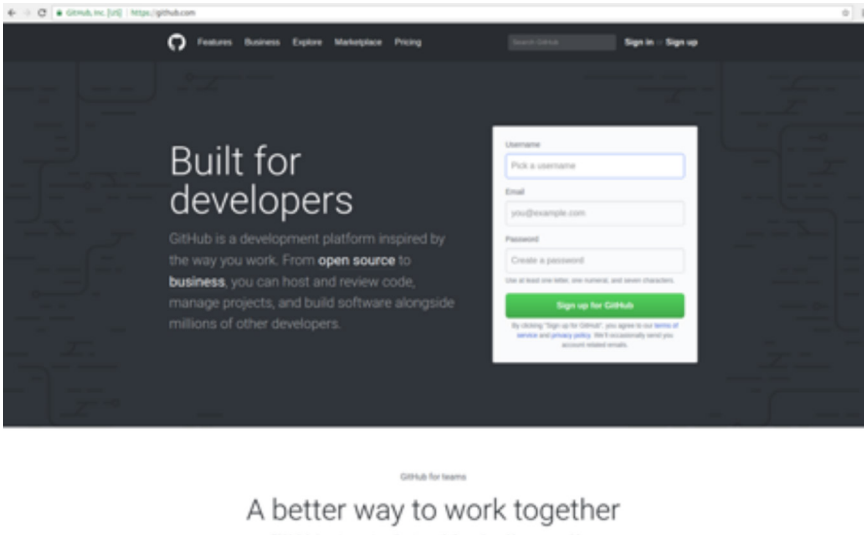

Figura 6. Plataforma GitHub.

Actualmente CodeAnalysis se encuentra accesible en la plataforma GitHub a través de un repositorio público con dirección https://github.com/KaizTen/code-analysis. En el repositorio se encuentra disponible el código fuente de la herramienta, ejemplos de códigos fuentes a evaluar, así como tests de unidad de ejemplo para aplicar. A su vez, en el repositorio se encuentran disponibles las instrucciones para llevar a cabo el proceso de compilación y de ejecución de la herramienta.

Figura 7. Repositorio de CodeAnalysis en la plataforma GitHub.

CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

En la actualidad, la programación informática constituye un campo de estudio de especial relevancia para el desarrollo de la sociedad de la información debido a sus numerosas aplicaciones prácticas. Al mismo tiempo, se trata de una materia transversal a múltiples áreas de conocimiento. Por estos motivos, la programación informática está recogida a día de hoy en los diversos planes de estudios de múltiples titulaciones universitarias de todo el mundo. Éste es el caso dentro de la Universidad de La Laguna del Grado en Ingeniería Informática, el Grado en Ingeniería Electrónica o el Grado en Matemáticas, entre otros. Esta situación se presenta también con asiduidad en otras instituciones universitarias españolas. En dichas titulaciones el alumnado se enfrenta a múltiples materias en las que tiene que demostrar sus conocimientos y habilidades de programación mediante la presentación y defensa de supuestos prácticos en forma de aplicaciones informáticas de limitado alcance y que deben adecuarse fielmente a las especificaciones técnicas planteadas para que sean consideradas como válidas por parte del profesorado.

Durante los últimos años, se ha evidenciado una falta notable en la disponibilidad de herramientas software que ayuden al alumnado y profesorado en el proceso de enseñanza-aprendizaje de materias asociadas a la programación informática. Entre los parámetros que mayor impacto tienen en el proceso de autoevaluación por parte del alumnado así como de corrección de prácticas por parte del profesorado se encuentran la disponibilidad temporal, el acceso a recursos didácticos y la facilidad de acceso a ejemplos interactivos. Estos parámetros son incluidos de forma natural en la herramienta software desarrollada en el presente trabajo. De esta manera se consigue que el tiempo de evaluación se reduzca notablemente. El motivo se encuentra en la automatización de pruebas y al mismo tiempo que se le proporciona al alumnado los recursos didácticos asociados a la materia y ejemplos representativos que faciliten la evaluación.

Desafortunadamente, la notable limitación temporal y de recursos humanos existente para la realización de las sesiones prácticas de programación informática hacen que aspectos como la corrección minuciosa o la inspección del buen uso de prácticas de programación, utilización de estructuras de datos o algoritmos requeridos no sea siempre tenga el nivel de detalle deseado. En este contexto, la incorporación de herramientas de las nuevas tecnologías de la información y las comunicaciones destinadas a la corrección de prácticas de informática por parte del profesorado y la autoevaluación del alumnado hace que se obtenga un proceso de enseñanza-aprendizaje más equitativo, eficaz y robusto que mediante los actuales métodos de evaluación.

El presente trabajo ha perseguido cumplir con un doble objetivo en base a la discusión anterior. En primer lugar, proporcionar un conjunto

de herramientas TIC que puedan ser progresivamente incorporadas en las materias con contenidos relativos a la programación informática en el marco de la Universidad de La Laguna y que permitan la corrección semi-automática por parte del profesor y, por otro lado, brindar al alumnado elementos de referencia para su proceso de autoevaluación en el contexto de las materias con contenidos de programación informática.

Finalmente, cabe destacar que la realización de este trabajo abre un amplio abanico de líneas de trabajo futuras. Algunas de las más interesantes son las descritas a continuación:

Validación de la herramienta creada a través de una experiencia amplia y heterogénea que permita evaluar los aspectos mejorables de la misma así como identificar aquellos elementos a incorporar en el futuro.

Creación de una plataforma web que permita tanto alumnos como a profesores poder realizar la ejecución y corrección de las prácticas de una forma gráfica. Esta plataforma podría ser un entorno que permita el acceso a históricos de evaluación por parte de los alumnos así como también un lugar común para compartir tests de unidad entre profesores de materias con contenidos comunes.

Creación de un módulo de detección de plagios que permita identificar aquellas entregas realizadas por parte de los alumnos que vulneran la autoría de las mismas. La incorporación de un módulo de propósito específico en este sentido permitiría ayudar en la detección de material duplicado a la vez que reduciría el tiempo de evaluación por parte del profesorado y aumentaría el nivel de desempeño por parte del alumnado.

REFERENCIAS BIBLIOGRÁFICAS

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. In *Software Engineering and Advanced Applications (SEAA), 39th EUROMICRO Conference on* (pp. 9-16). IEEE.

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. Computer science education, 15(2), 83-102.

Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley professional.

Brown, S., Race, P., & Smith, B. (2004). *500 tips on assessment*. Routledge.

Dennis, I., Newstead, S. E., & Wright, D. E. (1996). A new approach to exploring biases in educational assessment. *British Journal of Psychology*, *87*(4), 515-534.

Gómez, G. R., Saiz, M. S. I., y Jiménez, E. G. (2013). Autoevaluación, evaluación entre iguales y coevaluación: conceptualización y práctica en las universidades españolas. *Revista de investigación en educación*, *2*(11), 198-210.

Gordillo, J. J. T., y Rodríguez, V. H. P. (2006). La rúbrica como instrumento pedagógico para la tutorización y evaluación de los aprendizajes en el foro online en educación superior. Pixel-Bit. Revista de Medios y Educación, (36), 141-149.

Howell, K. (2003). First computer languages. *Journal of Computing Sciences in Colleges*, *18*(4), 317-331.

Humphrey, W. S. (1997). What do we know about programming?. In *Papers presented at the seventh workshop on Empirical studies of programmers* (pp. 224-232). ACM.

Jácome, É. P. (2013). La rúbrica y la justifica en la evaluación. *Íkala: Revista de Lenguaje y Cultura*, *18*(3).

Jakovljevic, M. (2003). Concept mapping and appropriate instructional strategies in promoting programming skills of holistic learners. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* (pp. 308-315). South African Institute for Computer Scientists and Information Technologists.

Keengwe, J., Onchwari, G., & Wachira, P. (2008). The use of computer tools to support meaningful learning. *AACE journal*, *16*(1), 77-92.

Lance, C. E., Fisicaro, S. A., & Lapointe, J. A. (1990). An examination of negative halo error in ratings. *Educational and Psychological Measurement*, *50*(3), 545-554.

Lau, W. W., & Yuen, A. H. (2009). Exploring the effects of gender and learning styles on computer programming performance: implications for programming pedagogy. *British Journal of Educational Technology*, *40*(4), 696-712.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 41, 51-61.

Maheshwari, P. (1997, July). Teaching programming paradigms and languages for qualitative learning. In *Proceedings of the 2nd Australasian conference on Computer science education* (pp. 32-39). ACM.

Malik, M. A. (2000). Technical opinion: on the perils of programming. Communications of the ACM, 43(12), 95-97.

Noche, B. G., Serra, V. Q., Ruiz, M. G., e Ibáñez, J. C. (2017). La evaluación y retroalimentación electrónica entre iguales para la autorregulación y el aprendizaje estratégico en la universidad: la percepción del alumnado. *REDU. Revista de Docencia Universitaria*, *15*(1), 127-146.

Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE software*, *17*(4), 26-32.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137-172.

Runeson, P. (2006). A survey of unit testing practices. *IEEE software*, *23*(4), 22-29.

Sudkamp, T. A. (1988). *Languages and machines: an introduction to the theory of computer science*. Addison-Wesley Longman Publishing Co., Inc.

Valero, M. y Navarro, J. J. (2010). Una colección de metáforas para explicar (y entender) el EEES. In Actas de las XVI Jornadas de Enseñanza Universitaria de Informática (JENUI 2010), 293–300.

Wirth, N. (1976). *Algorithms + Data Structures = Programs Prentice-Hall Series in Automatic Computation*. Prentice Hall.