Plataforma web para retroalimentación automática en la docencia de ensamblador

Pablo Fuentes, Carmen Martínez, Enrique Vallejo, Esteban Stafford y José Luis Bosque [1]

Resumen— **Este artículo introduce TutorSpim, una plataforma web desarrollada en la Universidad de Cantabria para ayudar en el proceso de retroalimentación en asignaturas de Estructura de Computadores con prácticas de programación en ensamblador. Se describe la motivación docente que impulsó el desarrollo, así como la estructura de la herramienta, y su funcionamiento con un ejemplo concreto. La opinión de los alumnos es que esta herramienta ayuda al proceso de aprendizaje y aumenta la motivación.**

Palabras clave— **TutorSpim, autoevaluación, retroalimentación, Moodle, MIPS.**

I. Introducción

Uno de los componentes principales del proceso de evaluación continua es la necesidad de proporcionar retroalimentación, es decir hacer consciente al alumno del nivel del progreso conseguido, así como permitir la identificación de los problemas más comunes de aprendizaje para solucionarlos mediante actividades y organizar la recuperación. Sin embargo, el problema fundamental que surge con la retroalimentación es que frecuentemente tiene un fuerte impacto en la carga de trabajo del profesor, llegando a ser totalmente inabordable en grupos con un número de alumnos medio o alto. En ocasiones la reducción del número de alumnos por grupo no es viable por problemas relacionados con la falta de recursos, tanto humanos como materiales, y hay que buscar soluciones alternativas para poder mantener estos esquemas de evaluación. Esta situación se da en las prácticas de laboratorio de asignaturas del ámbito de Estructura y Organización de Computadores. En concreto, nos hemos centrado en la asignatura Introducción a los Computadores de 1º curso de Grado en Ingeniería Informática, que presenta la problemática antes descrita.

Para ayudar en la tarea de verificación de código existen tanto aplicaciones de calificación automática del código con fines docentes (pueden encontrarse una recopilación bastante buena en [1] y [2]) como sistemas de jueces online orientados principalmente a concursos de programación (como TopCoder [3], Uva [4] o jutge.org [5], si bien este último tiene también claras intenciones docentes). Una alternativa muy relacionada puede encontrarse en [6], pero para evaluación de prácticas de Matlab, no estando aún disponible la versión que evalúa prácticas en ensamblador. En general, no hemos encontrado ninguna aplicación que se adapte por completo a nuestras necesidades, especialmente en cuanto a la arquitectura MIPS empleada y al nivel de retroalimentación a aportar al alumno. Para subsanar estos problemas docentes, se ha desarrollado **TutorSpim**, una herramienta de apoyo en la tarea de retroalimentación de los alumnos orientada a las prácticas de laboratorio relacionadas con la programación en ensamblador.

Esta herramienta está desarrollada como un módulo de Moodle al que los alumnos pueden subir el código de sus prácticas. Con el código de cada alumno, el servidor realiza una batería de comprobaciones y determina si el código es correcto o no, de acuerdo a una serie de casos determinados de antemano por el procesador, a modo de test unitario. A continuación, proporciona la retroalimentación correspondiente al alumno, para que conozca rápidamente si su código es correcto, o en qué casos concretos falla y cómo debería afrontar la corrección.

El resto del artículo está organizado como sigue. La siguiente sección entra en detalle en los problemas docentes que han motivado el desarrollo de la herramienta. La sección III describe la implementación realizada en la herramienta y su modo de funcionamiento. Finalmente, la sección IV muestra el funcionamiento con un caso concreto de ejemplo, así como la opinión de los alumnos respecto a la herramienta al final de la asignatura.

II. Motivación docente

Uno de los pilares fundamentales de la reforma formativa que supone el Espacio Europeo de Educación Superior (EEES) para la mejora de los resultados académicos de los universitarios es la introducción de la **evaluación continua** como mecanismo para determinar el grado de cumplimiento de las competencias y resultados de aprendizaje adquiridos por los alumnos. El proceso de evaluación continua no es un mero cambio en el número y distribución temporal de las pruebas de evaluación de una asignatura. Por el contrario, debe verse como una parte importante del proceso de aprendizaje y debe contribuir significativamente al mismo.

Uno de los componentes principales de este proceso es la necesidad de proporcionar **retroalimentación**, es decir hacer consciente al alumno del nivel del progreso conseguido, así como permitir la identificación de los problemas más comunes de aprendizaje para solucionarlos mediante actividades y organizar la recuperación. Para que se cumplan correctamente los objetivos de la evaluación continua, la retroalimentación debe:

- Proporcionarse con frecuencia, desde el inicio hasta la conclusión de la asignatura.
- Proporcionar a los estudiantes información sobre los resultados de las pruebas y también sobre la posibilidad de mejorar para el futuro.

[1]Dpto. de Electrónica y Computadores, Universidad de Cantabria {nombre.apellido}@unican.es

- Informar sobre el planteamiento de las clases y actividades siguientes.
- Venir de diferentes perspectivas: de las reflexiones de los estudiantes sobre su propio trabajo, de las reflexiones de los compañeros sobre el trabajo de los otros y de los docentes mismos.

Sin embargo la retroalimentación frecuentemente tiene un fuerte impacto en la carga de trabajo del profesor, llegando a ser totalmente inabordable en grupos con un número de alumnos medio o alto. Esta situación se da en muchas de las prácticas de laboratorio de primeros cursos de ingenierías; en concreto nos hemos centrado en asignaturas de Estructura, Tecnología y Organización de Computadores. En este ámbito un 50 % de las horas de clase se dedican a la realización de prácticas de laboratorio y en ellas se desarrollan una gran cantidad de las competencias y/o resultados de aprendizaje que el profesor debe evaluar. Las prácticas de laboratorio de estas asignaturas tienen relación típicamente con el desarrollo de programas en lenguaje ensamblador.

Por ello, el objetivo principal que nos hemos planteado con TutorSpim ha sido el diseño, implementación y puesta en marcha de una plataforma web que permita realizar el proceso de autoevaluación y retroalimentación de las prácticas de laboratorio relacionadas con la realización de programas en ensamblador, de forma automática y sin la necesidad de intervención del profesor. Durante el curso 2010-11 hemos desarrollado la herramienta y la hemos puesto en marcha en Introducción a los Computadores de 1º curso de Grado en Ingeniería Informática, una de las asignaturas del ámbito citado. Esta asignatura ya se había impartido el curso anterior, luego las prácticas a realizar y los problemas que puede encontrar el docente en el laboratorio ya eran conocidos. Además, es una asignatura con un elevado número de alumnos por grupo de prácticas. En años posteriores, se prevé la adaptación de la herramienta a otras asignaturas.

La asignatura forma parte de la materia básica Fundamentos de Informática de la titulación. En esta asignatura se estudia cómo funciona un procesador real de tipo RISC como es el MIPS [7], buscando los siguientes resultados de aprendizaje:

- Conocer las características principales de las unidades funcionales del computador, así como sus principios de funcionamiento.
- Entender cómo se representan los datos y las instrucciones en la memoria de un computador.
- Comprender la relación que existe entre la estructura del computador y el repertorio de instrucciones de bajo nivel en el que se puede programar.
- Comprender el funcionamiento interno del computador y de las distintas fases de ejecución de las instrucciones.

Para ello, en la planificación de la asignatura se hace especial énfasis en la programación en lenguaje ensamblador, que es una de las mejores maneras de aprender el funcionamiento de un procesador. En la asignatura están proyectadas 12 sesiones de prácticas de 2 horas cada una, divididas en 6 prácticas, para que el alumno domine la programación en ensamblador. En ellas se utiliza PCSpim [8], un simulador del procesador MIPS. Este simulador muestra al alumno los fallos de compilación (fallos en la sintaxis del programa) y el resultado de la ejecución en las diferentes unidades funcionales, pero no indica si el programa resuelve correctamente el problema propuesto. Queda por tanto, en manos del alumno y con la ayuda del profesor, probar el programa con diversos ejemplos o variaciones de los parámetros, para determinar la corrección del mismo. Durante estas sesiones el profesor ha de atender a 20 alumnos durante las 2 horas de duración, con una media de solo 6 minutos por alumno en cada sesión.

Ante esta situación, TutoSpim busca la autoevaluación por parte del alumno de las prácticas; proporcionar una mayor retroalimentación de la evaluación continua llevada a cabo en el laboratorio, y conseguir una mayor flexibilidad en la realización de las prácticas para los alumnos.

III. Descripción de la Herramienta

En esta sección se describe el diseño de la herramienta de evaluación automática TutorSpim. Para ello, comenzamos detallando los requerimientos de las prácticas del alumno y el enfoque de "caja negra" y análisis de declaración de variables empleado para validarlas. A continuación, describimos la implementación de la herramienta, tanto la parte del simulador como el frontend web. Finalmente, se abordan algunos aspectos de seguridad.

A. Modelo empleado para la corrección de prácticas

TutorSpim tiene como funcionalidad corregir el código de un alumno de acuerdo a unos requisitos que le ha pasado previamente el profesor. Dicho código estará escrito en lenguaje ensamblador del MIPS. Este código se divide en dos partes principales claramente diferenciadas: En primer lugar se declaran e inicializan las variables que se van a emplear; en segundo lugar aparecen las instrucciones que componen el programa.

A la hora de evaluar la corrección del código del alumno, se pueden emplear dos planteamientos opuestos. El modelo empleado tradicionalmente en el laboratorio es el análisis visual del código y de su ejecución en el simulador, verificando que las estructuras de datos y control empleadas sean correctas y apropiadas a los requisitos de la práctica. Este modelo podemos denominarlo de "caja blanca" (o transparente). Por el contrario, el modelo de "caja negra" se basa en ignorar por completo la implementación del código, y fijarse únicamente en la corrección de la salida que se obtiene para una entrada dada, o un conjunto de entradas que se correspondan con casos extremos. Este modelo de caja negra es el típicamente empleado por las herramientas de test unitario.

La verificación automática del código de un

alumno mediante un modelo de caja blanca es, en casi cualquier caso, imposible. Esto se debe a que los alumnos pueden realizar códigos diferentes que resuelvan correctamente el mismo problema, variando tanto las variables y registros empleados como las instrucciones y estructuras de control utilizadas. Por ello, TutorSpim no analiza la corrección del código mediante un análisis del mismo (caja blanca), sino mediante un modelo de caja negra. Para ello, el profesor fuerza un cierto conjunto de parámetros de entrada, y se comprueba si la ejecución del código devuelve el resultado esperado. Sin embargo, sí que se permite un cierto análisis del código que puede resultar interesante antes de la ejecución del mismo: se dispone de una serie de herramientas para verificar que se hayan definido las variables necesarias, y con el tipo de dato apropiado.

El problema del modelo de caja negra es cómo interactuar con el código del alumno, es decir, qué emplear como parámetros de entrada y salida. En general, existen dos opciones para fijar los parámetros de entrada en un simulador: bien mediante el valor de un registro o una variable en memoria (en el propio código del alumno) o bien mediante la entrada por consola. Igualmente, la salida puede mostrarse por la consola, o evaluarse de acuerdo al valor de un registro o una variable en memoria. La siguiente sección detalla cómo se ha implementado cada una de estas alternativas en TutorSpim, y cómo se comprueba la validez del código del alumno.

B. Simulador de máquina MIPS

Siguiendo el modelo planteado en la sección anterior, se han diseñado dos etapas en el proceso de validación. La primera etapa es la única que analiza el código del alumno, en concreto determinando si las variables declaradas e inicializadas cumplen los requerimientos exigidos por el profesor, como tipo de dato y tamaño reservado en memoria. Adicionalmente, en esta fase se puede reescribir el código del alumno, sustituyendo las declaraciones empleadas por otras diferentes. Esto permite emplear ciertas variables en memoria como parámetros de entrada para el modelo de "caja negra". Todos estos pasos se realizan en función del nombre de cada variable; evidentemente, para ello es necesario que el alumno emplee nombres de variables predefinidos, que el profesor le indica en el enunciado de la práctica.

Para validar las declaraciones de variables se ha empleado un análisis del texto mediante las herramientas 'grep' y 'sed' [9]. Estas herramientas permiten hacer búsquedas con expresiones regulares y reemplazarlas si es necesario. Para facilitar el trabajo del docente se han elaborado una serie de funciones que realizan las búsquedas y sustituciones. Cada función está asociada a un tipo de comprobación: existencia de una variable, tipo de variable empleada, valor inicial, y tamaño en memoria de una variable de tipo string. También permiten sustituir la declaración del alumno por otra distinta, para emplear el modelo de parámetros de entrada por memoria.

La segunda etapa verifica el programa escrito por el alumno mediante el modelo de caja negra, simulando la ejecución mediante la herramienta SPIM. SPIM es un simulador de máquina MIPS desarrollado con fines educativos y divulgativos [8]. Se ha elegido dicho simulador de entre los disponibles por tratarse de un programa de código abierto y que dispone tanto de interfaz gráfica (PCSpim/QtSpim) como de línea de comandos (xspim). Mediante la línea de comandos se puede automatizar una secuencia de operaciones a realizar para cargar y ejecutar el código del alumno. Esta serie de operaciones incluye tanto la entrada de parámetros por consola, como la comprobación del resultado del código, tanto por la salida estándar como leyendo registros del procesador o posiciones en memoria.

Dado que el programa puede ser interactivo y que código del alumno puede contener diversos tipos de errores, es necesario poder controlar la simulación y detenerla si el resultado no se ajusta a las especificaciones, reportando el correspondiente mensaje de error. Para interactuar con el simulador, especialmente a través de la consola, es necesaria una herramienta capaz de interpretar la salida estándar y de introducir comandos por la misma. Para ello se ha empleado el comando "expect", una extensión del lenguaje de *script* Tcl [10].

El lenguaje Tcl es relativamente complejo, por lo que al igual que en el caso de la verificación de las variables, se han desarrollado una serie de funciones de ayuda: introducir una entrada por consola, comprobar que la salida por consola es la esperada, y leer el valor de una posición en memoria o de un registro. Mediante dichas funciones en el orden apropiado, podemos asegurar que el comportamiento del programa responde a los requisitos fijados.

Para automatizar todas las modificaciones de parámetros de entrada y las comprobaciones de los parámetros de salida, las funciones explicadas anteriormente deben ser utilizadas desde un *script* de línea de comandos (concretamente del tipo *shell Bourne*). Será tarea del profesor elaborar dicho *script*, al que denominaremos *checkfile*, en base a una plantilla, empleando las funciones que precise. La figura 1 muestra un modelo de checkfile, que se explicará con más detalle mediante un caso práctico en la sección IV.

C. Interfaz web

El conjunto simulador de máquina MIPS más ficheros de corrección (*checkfiles*) nos permite corregir el código de un alumno de forma rápida y sencilla, pero es necesario dotar al sistema de una interfaz para que el propio alumno sea el que compruebe si su trabajo se ajusta a los requisitos.

Para la implementación se ha optado por partir de la plataforma de aprendizaje Moodle [11]. Al funcionar bajo una licencia de código abierto, nos permite desarrollar una interfaz específica para nuestro sistema y aprovechar las capacidades que ya están desarrolladas (autenticación de usuarios, asignación

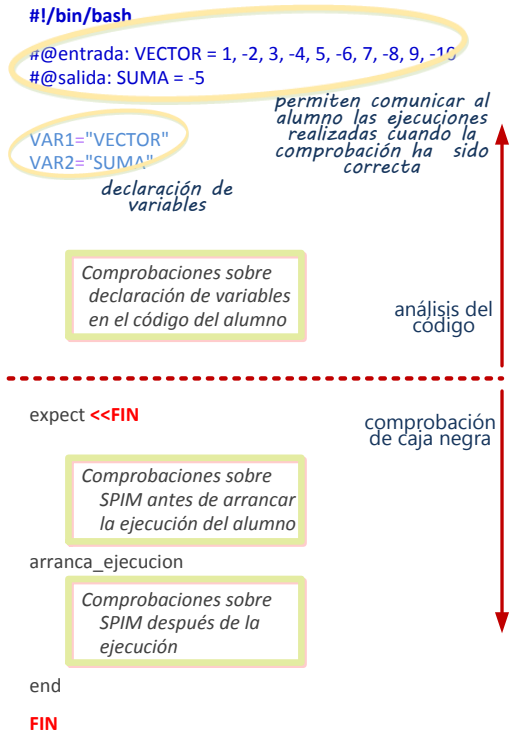

Fig. 1. Estructura de un checkfile

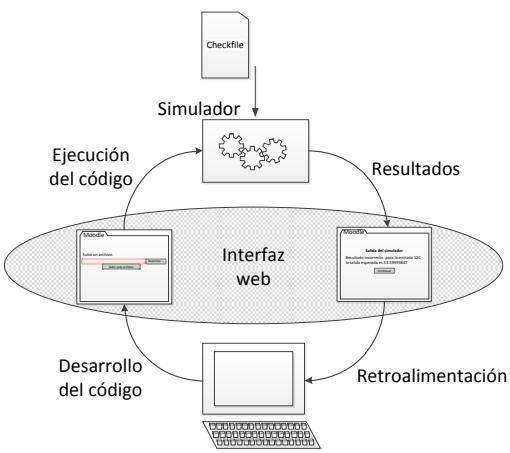

Fig. 2. Esquema de Simulación

de tareas, foro, descarga de ficheros...). Tutorspim está implementado como un módulo de Moodle programado en PHP [12], que nos permite ejecutar código en el servidor y ofrecer una respuesta dinámica. La estructura general es la mostrada en la Figura 2.

Moodle dispone por defecto de diferentes tipos de "tareas" para plantear al alumno. El módulo TutorSpim define un nuevo tipo de tarea. Moodle está configurado para tener 3 apariencias distintas, dependiendo del tipo de usuario que accede al sistema: edición, usuario y profesor. La primera vista es la de edición, que emplea el docente para crear una práctica nueva especificando el título, enunciado, fecha límite de entrega y otros parámetros similares. Aquí es donde el profesor ha de subir los *checkfiles* para corregir el código del alumno. Se pueden subir hasta 99 ficheros de corrección diferentes, para verificar el funcionamiento del programa en diferentes escenarios.

Una vez creada la tarea, el alumno puede acceder a la vista de usuario desde la que se le mostrará el título y enunciado de la tarea. En esta vista se permite subir un fichero que contiene el código ensamblador pedido en la práctica. Al subir el fichero, se ejecuta en el servidor una llamada al simulador con las condiciones fijadas por el profesor en cada uno de los *checkfiles*, simulando el código del alumno con diferentes parámetros de entrada. Tras finalizar todas las comprobaciones, se muestra una página de resultados que incluye un mensaje de error si el código no era correcto en algún caso. Si la ejecución es correcta, se le indica al alumno junto con un listado de todos los escenarios (los parámetros de entrada empleados y salidas obtenidas) en que se ha comprobado el funcionamiento.

Finalmente, el profesor puede acceder a una vista específica desde la que se presentan estadísticas de acceso y ejecución de los alumnos. También permite descargar las estadísticas y el código de cada alumno, distinguiendo además el último fichero subido con solución correcta.

D. Aspectos de seguridad

Permitir la ejecución de código en un servidor abre numerosos frentes de ataque; por este motivo, la seguridad debe estar ampliamente contemplada. A continuación se detallan algunos problemas potenciales, y cómo se han afrontado en la implementación actual de TutorSpim, así como líneas pendientes a mejorar.

- Carga del servidor: las simulaciones que se ejecutan en el servidor están limitadas a un tiempo máximo de 10 segundos; esto evita problemas con bucles infinitos, frecuentes por malicia o accidente.
- Alumnos hostiles: un alumno podría generar un código malicioso que bloquease el servidor o que accediese a otras partes del mismo para realizar acciones no permitidas. En este aspecto, se han tomado varias medidas. En primer lugar, si el código falla la fase de compilación, no se continúa la ejecución. En segundo lugar, está el *timeout* comentado en el punto anterio. Finalmente, el simulador corre en una "jaula" chroot, para evitar que tenga acceso a otras partes del sistema.
- Profesores hostiles: el checkfile es un script de shell que corre nativamente en el servidor. Por este motivo, un script mal definido puede bloquear el sistema. También, un profesor con intenciones maliciosas podría ejecutar cualquier código en el servidor. En este primer año de implementación no se ha tenido en cuenta este problema, especialmente porque la profesora responsable forma parte del equipo de desarrollo de TutorSpim. Una implementación general

requiriría de una validación del checkpoint antes de su ejecución.

Debido al último problema de seguridad relativo a la definición de los checkpoints, el servidor en el que se ha implementado TutorSpim no es el servidor Moodle general de la Universidad, sino un servidor propio ubicado en un entorno cerrado, y accesible únicamente desde las aulas de la Facultad.

IV. Un Caso Práctico

En esta Sección describimos parte de la última práctica del curso y el uso de la herramienta Tutorspim. Dicha práctica propone la realización de diferentes programas en ensamblador del MIPS que involucran aritmética flotante. En este estudio concreto, se propuso al alumno realizar un programa en horario de clase, seguidamente los alumnos comprobaron sus programas con Tutorspim y al finalizar 42 de ellos rellenaron una encuesta de satisfacción.

En la práctica que estudiamos se pedía realizar un programa en ensamblador del MIPS que:

- Pida al usuario una temperatura.
- Pregunta en qué unidades está, CELSIUS o FAHRENHEIT.
- Si el usuario introduce el caracter F, muestra por consola la equivalencia de la temperatura en Celsius y finaliza el programa.
- Si el usuario introduce el caracter C, muestra por consola la equivalencia de la temperatura en Fahrenheit y finaliza el programa.
- Si se introduce algo diferente de F ó C, retorna el mensaje "ERROR" y finaliza el programa.

Para la corrección de este sencillo programa, el diseño del checkfile es como sigue. Primeramente, se tendrán que definir los parámetros de la ejecución. Puesto que el programa sólo necesitará dos parámetros de entrada (temperatura y unidades) tendremos que definir estos dos. En las siguientes líneas de código se puede observar que el programa del alumno se va a ejecutar dando como entrada el valor 12 y como unidades el carácter C, correspondiente a unidades Celsius.

```
# Parametros
PAR_TEMPERATURA=12
PAR_UNIDADES=C
```

Además, se definen como parámetros de salida tanto el valor en flotante que debe retornar después de la conversión, como el mensaje de error que se espera en caso de que la entrada de datos sea erróna:

```
PAR_MENSAJE="ERROR"
PAR_SALIDA=53.59999847
```

Por otro lado, los parámetros de salida esperados son los valores que se corresponden con la conversión correcta, esto es:

```
#@entrada: 12 C
#@salida: 53.59999847 F
```

Todos estos parámetros de entrada y salida se leen y escriben mediante la consola; la modificación de variables o registros se había empleado en prácticas anteriores. En esta práctica no se realiza análisis de "caja blanca" ya que no hay ningún requisito sobre tipos de variables a emplear; más bien, dicho requisito es implícito para lograr la solución correcta. Según esto, cuando se lanza la ejecución, se introducen los parámetros de entrada anteriores:

```
entrada $PAR_TEMPERATURA
entrada $PAR_UNIDADES
```

Para la comprobación de la salida, el parámetro a corregir es `$PAR_SALIDA`. Si la comprobación es incorrecta, la retroalimentación retornará tanto la entrada empleada como la salida esperada:

```
corrige $PAR_SALIDA "Para la entrada
$PAR_TEMPERATURA $PAR_UNIDADES la salida
esperada es $PAR_SALIDA \n"
```

Para la comprobación de este programa se prepararon 5 checkfiles como el descrito anteriormente con diferentes parámetros de entrada. Cada uno de ellos se corresponde con la ejecución con diferentes datos de entrada, incluyendo la posibilidad de que el programa tenga que retornar el mensaje de error que se solicita en el enunciado de la práctica. Por tanto, una ejecución correcta genera un mensaje como el que se puede ver en la Figura 3.

Fig. 3. Mensaje Tutorspim Ejecución Correcta

En caso de que la ejecución sea incorrecta porque el valor que muestra el programa no es el esperado, la herramienta muestra el mensaje que se puede ver en la Figura 4.

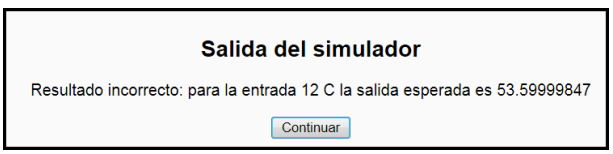

Fig. 4. Mensaje Tutorspim Ejecución Incorrecta

Una vez realizada la práctica, se repartió entre los alumnos una sencilla encuesta de satisfacción, de las cuales se recogieron un total de 42 encuestas. En dicha encuesta se realizan las preguntas:

1. Di una cosa que hace bien TutorSpim

2. Di una cosa que hace mal TutorSpim
3. Di una cosa que te gustaría que hiciese

Con respecto a los aspectos erróneos de la herramienta, los alumnos señalaron principalmente que Tutorspim es muy estricto en cómo tienen se tienen que retornar los resultados para la comprobación (en particular, el uso de mayúsculas/minúsculas), y la complejidad del proceso de autenticación al no estar integrado con el servidor Moodle principal de la Universidad.

Por otro lado, como característica deseable de Tutorspim, se propusieron entre otras que diese algo más de información cuando las ejecuciones son erróneas y que se pudiese utilizar fuera de los laboratorios del centro mediante acceso web. Estas ideas quedan pendientes para revisiones posteriores.

Como aspectos positivos los alumnos resaltaron que hace bien lo que tiene que hacer (comprobar los programas con diferentes datos) y que ha funcionado correctamente en la mayoría de los casos. Sorprendentemente, uno de los aspectos que resaltaron varios alumnos fue el hecho de que obtener un mensaje declarando correcta la ejecución tiene un efecto motivador en el alumno. Además de las cuestiones anteriores, se solicitó que valorasen de 1 a 5 los siguientes aspectos:

- Facilidad de uso
- Utilidad de la información
- Claridad del mensaje retornado

Los resultados de las mismas se han resumido en la gráfica que se muestra en la Figura 5. Como se puede observar, los tres aspectos encuestados de la herramienta Tutorspim han obtenido una puntuación razonable, lo que nos anima a continuar con el desarrollo de la misma.

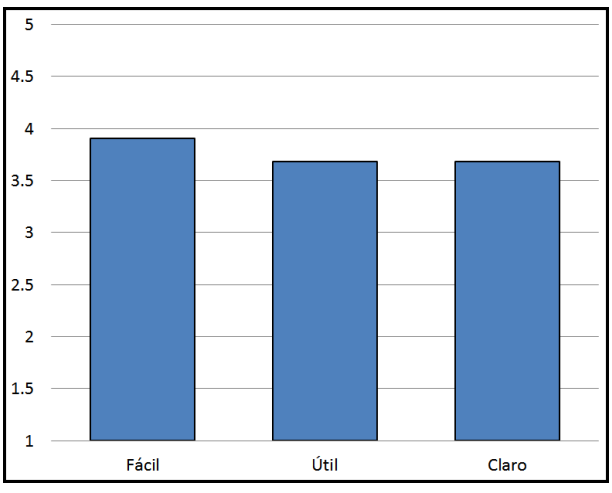

Fig. 5. Resultados Numéricos Encuesta Satisfacción

V. Conclusiones y líneas futuras

Respecto a la experiencia con la herramienta, nos gustaría destacar dos aspectos positivos que nos impulsan a seguir con su desarrollo en el futuro. Por una parte, la buena acogida por parte de los alumnos, como se ha podido ver en la sección anterior. Tanto la puntuación de las encuestas de satisfacción como las diferentes sugerencias por parte de los alumnos nos hacen pensar que continuar con el desarrollo de la herramienta implicará un mayor uso por parte de los alumnos. Por otra parte, nuestra experiencia personal con la herramienta ha sido positiva y consideramos que se tiene que seguir con ella. Además, después de trabajar con la herramienta consideramos que el enfoque inicial de ayuda en las sesiones de prácticas ha sido el correcto, pero que se podría extender a otros ámbitos; en concreto, la posibilidad de la corrección y evaluación automática de las prácticas en el laboratorio utilizando TutorSpim.

Agradecimientos

Los autores quieren agradecer a Emilio Castillo y Cristóbal Camarero su ayuda. Este trabajo ha sido posible gracias a un Proyecto de Innovación Docente promovido y financiado por el Vicerrectorado de Calidad e Innovación Educativa de la Universidad de Cantabria; por el Ministerio de Ciencia e Innovación bajo el proyecto TIN2010-21291-C02-02, la red de excelencia HiPEAC y el proyecto Consolider Supercomputación y e-Ciencia.

Referencias

[1] Christopher Douce, David Livingstone, and James Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, no. 3, Sept. 2005.

[2] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, New York, NY, USA, 2010, Koli Calling '10, pp. 86–93, ACM.

[3] Nikolay Archak, "Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder.com," in *Proceedings of the 19th international conference on World wide web*, New York, NY, USA, 2010, WWW '10, pp. 21–30, ACM.

[4] M.A. Revilla, S. Manzoor, and R. Liu, "Competitive learning in informatics: The UVa online judge experience," *Olympiads in Informatics*, vol. 2, pp. 131–148, 2008.

[5] J. Petit Silvestre and S. Roura Ferret, "Programación-1: Una asignatura orientada a la resolución de problemas," *Jornadas de Enseñanza Universitaria de la Informática (15es: 2009: Barcelona)*, 2009.

[6] J. Ramos, M.A. Trenas, E. Gutiérrez, and S. Romero, "E-assessment of Matlab assignments in Moodle: Application to an introductory programming course for engineers," *Computer Applications in Engineering Education*, 2011.

[7] John Hennessy, Norman Jouppi, Steven Przybylski, Christopher Rowen, Thomas Gross, Forest Baskett, and John Gill, "Mips: A microprocessor architecture," *SIGMICRO Newsl.*, vol. 13, no. 4, pp. 17–22, Oct. 1982.

[8] J.R. Larus, *SPIM S20: A MIPS R2000 simulator*, Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin, 1990.

[9] L.E. McMahon, "Sed, a non-interactive text editor," *UNIX Programmer's Manual*, vol. 2, 1978.

[10] D. Libes, "expect: Scripts for controlling interactive processes," *Computing Systems*, vol. 4, no. 2, pp. 99–125, 1991.

[11] M. Dougiamas and P. Taylor, "Moodle: Using learning communities to create an open source course management system," in *Proceedings of world conference on educational multimedia, hypermedia and telecommunications*, 2003, vol. 3.

[12] M.E. Davis and J.A. Phillips, *Learning PHP and MySQL*, O'Reilly Media, Inc., 2006.