MSEL
Modelling in Science Education and Learning

Modelling in Science Education and Learning
Volume 16 (2), 2023 doi: 10.4995/msel.2023.19256.
Instituto Universitario de Matemática Pura y Aplicada
Universitat Politècnica de València

Implementación de casos de prueba negativos en actividades VPL de Moodle

Development of negative test cases in Moodle VPL activities

Pablo Carmona del Barco, J. Álvaro Fernández Muñoz, José M. Perea Ortega
Universidad de Extremadura
pablo@unex.es, jalvarof@unex.es, jmperea@unex.es

Abstract

VPL (Virtual Programming Lab) es un plugin de Moodle que permite automatizar la evaluación del código de un programa escrito en un determinado lenguaje de programación. Actualmente, VPL solo permite configurar casos de prueba positivos durante la evaluación, es decir, posibilita mostrar un mensaje de error al estudiante cuando la salida del programa difiere de la esperada para cada caso. Esto limita la identificación del tipo de error cometido y su consecuente retroalimentación, ya que podrÁan existir problemas para los que sea conveniente comprobar si se obtiene una salida especÁfica que no deberÁa darse y que estarÁa motivada por algún error que se está cometiendo en el programa. Este trabajo presenta una mejora para paliar esta limitación que consiste en incluir casos de prueba negativos en el proceso de evaluación de VPL, de manera que también se pueda mostrar un mensaje de error cuando la salida del programa coincida con la configurada para cada uno de los errores previstos en el problema a resolver. Dicha mejora facilita al docente la identificación de errores previsibles en el código que entrega el estudiante y permite, en cierto modo, orientarle hacia el tipo de error cometido sin que este conozca el caso de prueba que lo provocó.

VPL (Virtual Programming Lab) is a Moodle plugin that automates the code evaluation of a program written in a given programming language. Currently, VPL only allows the configuration of positive test cases during the evaluation, i.e. it makes it possible to display an error message to the student when the output of the program differs from the expected output for each case. This limits the identification of the type of error committed and its consequent feedback, since there could be problems for which it is necessary to check if a specific output is obtained that should not be given and that would be motivated by some logical error that is not being controlled in the program. This work presents an improvement to alleviate this limitation, which consists of including negative test cases in the VPL evaluation process, so that an error message can also be displayed when the output of the program matches the one configured for each of the errors expected in the problem to be solved. This improvement makes it easier for the teacher to identify potential errors in the code delivered by the student and allows, in some way, to guide the student towards the type of error committed without the student knowing the test case that caused it.

Palabras clave: Laboratorios de Programación Virtuales, LPV, Moodle, evaluación automática
Keywords: Virtual Programming Lab, VPL, Moodle, automatic evaluation

1. Introducción

En el campo de laí Informática, la programación es el proceso de crear programas mediante el desarrollo de un código escrito con algún lenguaje de programación, código que determina qué acciones debe realizar el ordenador para resolver un problema. La enseñanza y el aprendizaje de ese proceso de manera exitosa continúa siendo un reto hoy día. En la bibliografía se pueden encontrar diferentes propuestas para abordar ese desafío, una de las cuales consiste en evaluar la solución proporcionada por el código desarrollado a partir de una batería de casos preestablecidos de entrada/salida, comparando el resultado esperado con el obtenido por el programa para cada caso (Berssanette y De Francisco, 2021; Cheah, 2020).

Existen diversas herramientas disponibles que automatizan la evaluación de programas de ordenador que han sido escritos utilizando un determinado lenguaje de programación (Cardoso, Vieira, y Rocha, 2018). Una de ellas es VPL (*Virtual Programming Lab*)[1], una iniciativa desarrollada para la plataforma de gestión de aprendizaje Moodle que permite editar y ejecutar programas y habilitar la evaluación automática y continua (Rodríguez del Pino, Rubio Royo, y Hernández Figueroa, 2010, 2012). VPL admite el uso de varios lenguajes de programación e incorpora pruebas automáticas de caja negra, siguiendo un enfoque de entrada/salida. Este tipo de herramientas ayudan a reducir la sobrecarga del docente durante la evaluación de los estudiantes y mejoran la motivación y retroalimentación de estos durante su proceso de aprendizaje.

Por defecto, la evaluación de una actividad VPL en Moodle muestra, para cada caso no superado, o bien la entrada utilizada, la salida proporcionada por el programa y la salida esperada (lo que desvelaría al estudiante el caso concreto que ha fallado), o bien un mensaje de error que describa el error cometido. Dado que este mensaje supone ocultar la entrada/salida de los casos fallidos, a menudo el profesor opta por incluirlo con el único objetivo de no desvelar el caso fallido, pero empleando un texto genérico del tipo *Caso incorrecto*, que no aporta información sobre el tipo de error cometido, debido a que identificar el error concreto cometido es prácticamente imposible si se emplean solo casos *positivos*, es decir, casos considerados fallidos cuando la salida del programa propuesto difiere de la esperada. Para ilustrar este inconveniente, supóngase un problema sencillo consistente en leer desde teclado dos valores enteros mayores que 0 y mostrar como salida el ancho del intervalo cerrado que ambos definen (Algoritmo 6.1).

```
1    leer(a,b)
2    mientras (a<=0 o b<=0):
3        leer(a,b)
4    mostrar(|b-a|+1)
```

Listing 6.1: Pseudocódigo para calcular el ancho de un intervalo cerrado

Supóngase también la siguiente relación de errores previsibles:

1. Admitir la lectura de algún valor no mayor que 0.
2. Asumir que siempre se cumple $a <= b$ (solución errónea: $b - a + 1$).
3. Calcular el ancho del intervalo abierto (solución errónea: $|b - a| - 1$).
4. Calcular el ancho como la distancia entre a y b (solución errónea: $|b - a|$).

[1] http://vpl.dis.ulpgc.es

La Tabla 1 muestra una posible batería de casos positivos para detectar esos errores. Para cada caso se muestran los valores de entrada, la salida esperada, el error que se pretende identificar, y el mensaje que recibirá el estudiante si su salida no es la esperada. Puede observarse que el caso 3 unifica los errores 3 y 4 al ser imposible discernir entre ellos, y, por ello, se proporciona el mensaje genérico *Cálculo incorrecto*. Además, si se cometen los errores 3 o 4, la evaluación es errónea para los 4 casos, pues ninguno proporciona la salida esperada, recibiendo el estudiante mensajes sobre errores no cometidos. Esto se ilustra en las celdas sombreadas de la Tabla 2, donde se recopilan, para cada error, los casos positivos exitosos y fallidos.

Caso	Entrada	Salida	Error	Mensaje de error
1	0 7 2 7	6	1	Los valores deben ser mayores que 0
2	7 2	6	2	El primer valor puede ser mayor que el segundo
3	2 7	6	3,4	Cálculo incorrecto

Tabla 1: Ejemplo de batería de casos positivos para el problema del ancho del intervalo cerrado

Caso	Error cometido			
	(1)	(2)	(3)	(4)
1	X	✓	X	X
2	✓	X	X	X
3	✓	✓	X	X

Tabla 2: Relación entre el error cometido y los casos positivos exitosos (✓) y fallidos (X) para el problema del ancho del intervalo cerrado

En resumen, con el ejemplo se ilustran dos problemas que pueden surgir al contemplar únicamente casos positivos: por un lado, la imposibilidad de discernir entre ciertos errores distintos y, por otro, la posibilidad de proporcionar una retroalimentación inadecuada al estudiante mostrando mensajes sobre errores que no ha cometido. En este trabajo los autores proponen una mejora para mitigar estos problemas. Para ello, se han implementado una serie de desarrollos en el propio plugin de VPL que facilitan la identificación de errores previsibles y permiten, en cierto modo, orientar al estudiante hacia el tipo de error cometido sin conocer el caso de prueba que lo provocó. Esos desarrollos han consistido en establecer una relación directa entre una entrada y su salida errónea para cada uno de los errores previstos, integrando en la batería un nuevo tipo de casos que los autores han denominado *negativos*. De este modo, un caso negativo se considerará fallido si la salida del programa coincide con la indicada en el caso.

El resto de este trabajo continúa describiendo la metodología seguida en la Sección 2, mientras que la Sección 3 presenta los resultados. Por último, la Sección 4 resume las consideraciones finales y las propuestas para futuros trabajos.

2. Metodología

Dentro del marco de acciones de innovación docente promovidas por la Universidad de Extremadura (UEx) durante el curso 2020-21, los autores experimentaron el uso de VPL durante la enseñanza de contenidos prácticos en asignaturas de programación básica. Para ello, se utilizó la plataforma de enseñanza virtual disponible en la UEx, que está implementada en Moodle[2], en la que se diseñaron actividades VPL para apoyar el proceso de enseñanza-aprendizaje de los estudiantes de esas asignaturas.

[2] http://campusvirtual.unex.es

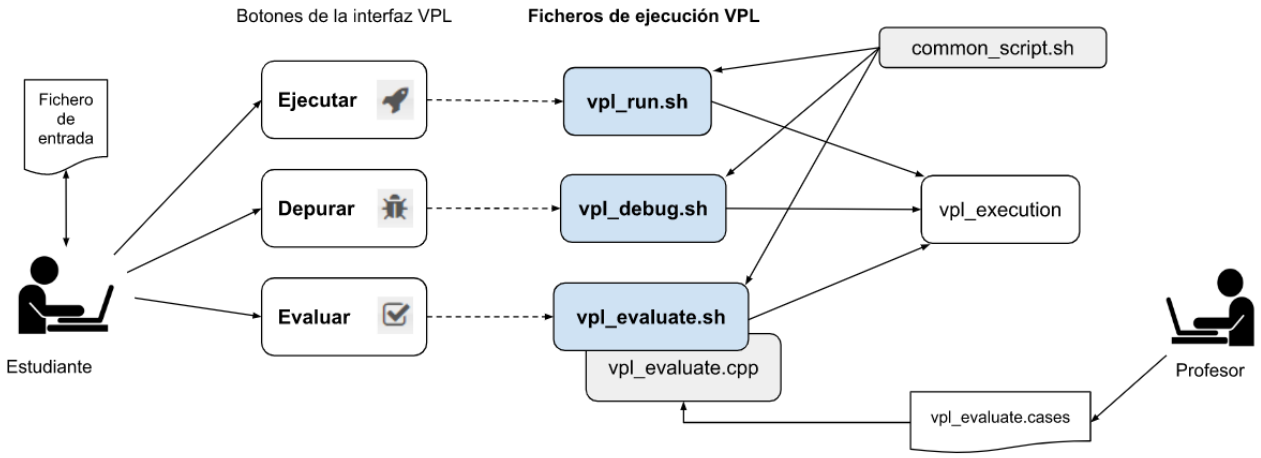

Figura 1: Modelo de ejecución de VPL

Para evaluar cualquier solución proporcionada por los estudiantes, VPL emplea el modelo de ejecución que se muestra en la Figura 1, donde existe un conjunto de archivos que participan en las tareas de ejecución, depuración y evaluación. Estos archivos pueden incluir secuencias de comandos, archivos de prueba de programas, archivos de datos, etc., e irán junto con los archivos enviados por el estudiante para ejecutarse en el servidor de ejecución. De entre ellos, los archivos `vpl_run.sh`, `vpl_debug.sh`, y `vpl_evaluate.sh` son scripts que describen por defecto las tareas a realizar durante la ejecución, depuración y evaluación de la propuesta, respectivamente, generando un archivo llamado `vpl_execution` que es el que finalmente ejecuta el servidor VPL. Ahora bien, cualquiera de estas acciones por defecto puede ser sustituida por otra si el docente sobrescribe el contenido del archivo script correspondiente, lo que requiere tener cierto conocimiento del lenguaje *shell*.

Los scripts por defecto incluyen siempre al comienzo la llamada a un script auxiliar llamado `common_script.sh`, que define funciones auxiliares y variables de entorno del *shell* con información sobre la tarea VPL, tales como LANG, para especificar el lenguaje de programación usado, o VPL_SUBFILE#, para referenciar cada nombre de los archivos enviados por el estudiante, donde # varía desde 0 hasta $n-1$, siendo n el número total de archivos enviados por el estudiante. Por esta razón, cuando el docente sobrescribe alguno de los scripts por defecto para personalizar el comportamiento de la acción asociada, es habitual que incluya también una llamada a `common_script.sh`, para así disponer de los recursos que proporciona.

Otro aspecto a considerar tiene que ver con el procesamiento de los casos de prueba que se definen para evaluar la solución del estudiante. Para ello, el script por defecto `vpl_evaluate.sh` emplea a su vez un programa llamado `vpl_evaluate.cpp` que procesa los casos de prueba contenidos en el archivo `vpl_evaluate.cases`, que es donde el docente debe incluirlos empleando una sintaxis específica. La Figura 4 muestra un ejemplo de configuración del fichero de casos `vpl_evaluate.cases`, donde se han definido dos casos de prueba: en el primero no se especifica el mensaje de error que mostraría VPL si la salida esperada para resolver la ecuación de segundo grado $x^2 - 3x + 2$ no coincide con la proporcionada por la solución del estudiante, mientras que en el segundo se configura el mensaje *Solución incorrecta* a través de la variable `Fail message`.

vpl_evaluate.cases

```
1
2  Case = Resolución de la ecuación cuadrada x^2-3x+2
3  Grade reduction = 60
4  input = 1 -3 2
5  output = 1 2
6  output = 2 1
7
8  Case = Resolución de ecuación cuadrada
9  Fail message = Solución incorrecta
10 Grade reduction = 70%
11 input = 2 -5 6
12 output = 2 3
13 output = 3 2
```

Figura 2: Ejemplo de fichero `vpl_evaluate.cases`

3. Resultados

En este trabajo se propone utilizar el código mostrado en la Figura 5 como contenido alternativo del script `vpl_evaluate.sh`. Téngase en cuenta que dicho código está configurado para evaluar soluciones escritas en lenguaje Python por lo que, al ser un lenguaje interpretado, no es necesario una instrucción previa de compilación. Por tanto, para otros lenguajes de programación interpretados, solo habría que modificar la línea 19, que es en la que se ejecuta el intérprete del lenguaje (*python3* en nuestro caso), junto con el fichero que contiene el código que entrega el estudiante, identificado a través de la variable de entorno $VPL\_SUBFILE0$, que se activa al cargar el script auxiliar `common_script.sh`. En el caso de que el lenguaje fuera compilado (Java o C, entre otros), habría que añadir la instrucción de compilación antes del bucle *for* de la línea 18, para así compilar una única vez y no hacerlo cada vez que se evalúa un caso. Además, habrá que modificar la línea 19 con la ejecución del fichero ejecutable generado durante la compilación.

Ahora bien, aunque cuando se personaliza el archivo `vpl_evaluate.sh` lo habitual es integrar los casos de prueba en el propio archivo, como novedad en nuestra propuesta los casos de prueba se han situado en un archivo `vpl_evaluate.cases` independiente, de modo que el docente pueda especificar los casos de prueba de forma más sencilla y similar al procedimiento estándar, sin necesidad de editar el archivo `vpl_evaluate.sh` para cada nuevo problema. No obstante, dado que el procesamiento de los casos de prueba se hace mediante el archivo `vpl_evaluate.sh` personalizado de la Figura 5 y no mediante el archivo `vpl_evaluate.cpp` que emplea la evaluación por defecto, ha sido necesario modificar el formato de descripción de los casos de prueba. Como se puede observar, en nuestra propuesta se hace uso de diferentes variables que se configuran en el fichero `vpl_evaluate.cases` para cada caso de prueba establecido:

- `nCase`: contabiliza el número total de casos de prueba definidos.
- `Name[$nCase]`: especifica la descripción del caso de prueba.
- `FailMsg[$nCase]`: establece el mensaje de error asociado al caso.
- `Type[$nCase]`: permite establecer si el caso de prueba se considera positivo o negativo.
- `GradeRed[$nCase]`: especifica la penalización que afectará a la calificación final del estudiante si la salida de su programa coincide con la esperada (para casos negativos) o difiere de la esperada (para casos positivos).

vpl_evaluate.sh

```
#! /bin/bash

# cargar el fichero common script y comprobar si Python 3 existe
. common_script.sh
check_program python3

# establecer el contador del número de casos (nCase) a 0
echo "nCase=0" > vpl_execution

# cargar los casos de evaluación y actualizar el contador nCase
cat vpl_evaluate.cases >> vpl_execution

cat >> vpl_execution <<EOF
    testsPassed=0
    someFailedTest=0
    grade=${VPL_GRADEMAX%%.*}   # parte entera de VPL_GRADEMAX

    for (( i=1; i <= nCase; i++ )); do
        # ejecutar el programa del estudiante
        echo "\`python3 ${VPL_SUBFILE0} < data\${i}.in\`" > user.out
        # calcular la diferencia entre la salida del programa y la salida esperada
        diff -y -w --ignore-all-space user.out data\${i}.out > diff.out
        result=\$?
        if [[ \$result > 0 && \${Type[\${i}]} == "+" || \$result == 0 && \${Type[\${i}]} == "-" ]]; then
            if [[ \$someFailedTest == 0 ]]; then
                echo "Comment :=>>- == TESTS ERRÓNEOS =="
                echo "Comment :=>>"
                someFailedTest=1
            fi
            echo "Comment :=>>-Test \$i: \${Name[\$((i))]}"
            echo "Comment :=>> \${FailMsg[\$((i))]}"
            echo "Comment :=>> "
            grade=\$((grade-\${GradeRed[\$((i))]}))
        else
            testsPassed=\$((testsPassed+1))
        fi
    done
    echo "Comment :=>>- == Resumen de la evaluación =="
    echo "Comment :=>>> +--------------------+"
    echo "Comment :=>>> |  \$nCase casos evaluados / \$testsPassed casos superados |"
    echo "Comment :=>>> +--------------------+"
    if [[ grade -lt 0 ]]; then
        grade=0
    fi
    echo "Grade :=>> \$grade"
EOF

chmod +x vpl_execution

```

Figura 3: Ejemplo de fichero `vpl_evaluate.sh` para incluir casos de prueba negativos

Retomando el problema descrito en la Sección 1 que consistía en leer desde teclado dos valores enteros mayores que 0 y mostrar como salida el ancho del intervalo cerrado que ambos definen, la Tabla 3 muestra una posible batería de casos negativos para dicho problema, mientras que la Tabla 4 muestra la relación entre errores cometidos y casos negativos que se cumplen. Puede observarse que ahora sí sería posible identificar de manera independiente cada error previsto y, por lo tanto, orientar al estudiante de la forma adecuada.

Cabe señalar que, si bien los casos negativos resultan más adecuados para identificar errores previsibles, no bastan por sí solos para evaluar una solución. La batería de casos de prueba debería incluir siempre, además de negativos, al menos un caso positivo. De lo contrario, un programa que nada tenga que ver con el problema a resolver se consideraría correcto al satisfacer todos los casos, ya que no habría coincidencia entre la salida del programa y la configurada en cada caso negativo. La Figura 6 muestra un extracto del fichero de casos de prueba `vpl_evaluate.cases` establecido para el problema del ancho del intervalo cerrado, en el que aparece la configuración del caso positivo 3 (ver Tabla 1) y el caso negativo -1 (ver Tabla 3).

Caso	Entrada	Salida	Error	Mensaje de error
-1	0 7 1 100	8	1	Los valores deben ser mayores que 0
-2	7 2	-4	2	El primer valor puede ser mayor que el segundo
-3	2 7	4	3	Debe calcularse el ancho del intervalo cerrado, no abierto
-4	2 7	5	4	El intervalo cerrado debe incluir ambos extremos

Tabla 3: Ejemplo de batería de casos negativos para el problema del ancho del intervalo cerrado

Caso	Error cometido			
	(1)	(2)	(3)	(4)
-1	X	✓	✓	✓
-2	✓	X	✓	✓
-3	✓	✓	X	✓
-4	✓	✓	✓	X

Tabla 4: Relación entre el error cometido y los casos negativos exitosos (✓) y fallidos (X) para el problema del ancho del intervalo cerrado

```
nCase = $((nCase+1))
Name[$nCase] = "Caso 3: Comprobación del cálculo"
FailMsg[$nCase] = "Cálculo incorrecto"
Type[$nCase] = "+"
GradeRed[$nCase] = 25
cat > data${nCase}.in <<EOF
2 7
EOF
cat > data${nCase}.out <<EOF
6
EOF

nCase = $((nCase+1))
Name[$nCase] = "Caso -1: Control de entrada"
FailMsg[$nCase] = "Los valores deben ser mayores que 0"
Type[$nCase] = "-"
GradeRed[$nCase] = 25
cat > data${nCase}.in <<EOF
0 7 1 100
EOF
cat > data${nCase}.out <<EOF
8
EOF
```

Figura 4: Extracto del fichero `vpl_evaluate.cases` para el problema del ancho del intervalo cerrado

4. Conclusiones

Este trabajo presenta una mejora respecto al proceso de evaluación automática que utiliza por defecto el plugin VPL (*Virtual Programming Lab*) de Moodle. La versión actual de VPL (4.0.1) tiene el inconveniente de que solo permite configurar casos de prueba *positivos* al evaluar los programas que entregan los estudiantes, es decir, posibilita mostrar un mensaje de error cuando la salida del programa difiere de la esperada para cada caso. Esto limita considerablemente la identificación del tipo de error cometido y su consecuente retroalimentación, ya que podrían existir problemas para los que sea conveniente comprobar si se obtiene una salida específica que no debería darse y que estaría motivada por algún error que se está cometiendo en el programa. De hecho, esta limitación quedó patente tras una encuesta realizada a los estudiantes que participaron en una experiencia piloto de uso de VPL en asignaturas de programación básica. Esta experiencia se llevó a cabo dentro del marco de acciones de innovación docente promovidas por la Universidad de Extremadura durante el curso 2020-21 y reveló que uno de

los inconvenientes con el que se encontraron los estudiantes fue la dificultad para comprender el por qué de los errores cometidos, ya que VPL les proporcionaba escasa o ninguna orientación en cuanto al tipo de error en cada caso. Este hecho derivó a menudo en la frustración de los estudiantes, que abandonaban la actividad sin resolver el problema ni entender por qué su solución era errónea.

La propuesta presentada en este trabajo está relacionada con los ficheros `vpl_evaluate.sh` y `vpl_evaluate.cases` que proporciona el plugin de VPL. Los autores proponen la inclusión de casos de prueba *negativos* en el proceso de evaluación de VPL, de manera que también se pueda mostrar un mensaje de error cuando la salida del programa coincida con la configurada para cada uno de los errores previstos en el problema a resolver. Esta mejora facilita al docente la identificación de errores previsibles en el código que entrega el estudiante y permite, en cierto modo, orientarle hacia el tipo de error cometido sin que este conozca el caso de prueba que lo provocó.

Referencias

Amaya Berssanette, J. H., y De Francisco, A. C. (2021).
Active learning in the context of the teaching/learning of computer programming: A systematic review.
Journal of Information Technology Education: Research, 20.
doi:10.28945/4767

Cardoso, M., Vieira, A., y Rocha, A. (2018).
Integration of virtual programming lab in a process of teaching programming EduScrum based.
En Iberian Conference on Information Systems and Technologies, CISTI.
doi:10.23919/CISTI.2018.8399261

Cheah, C. S. (2020).
Factors contributing to the difficulties in teaching and learning of computer programming: A literature review.
Contemporary Educational Technology, 12.
doi:10.30935/cedtech/8247

Rodríguez del Pino, J. C., Rubio Royo, E., y Hernández Figueroa, Z. (2010).
VPL: laboratorio virtual de programación para Moodle.
En XVI Jornadas de Enseñanza Universitaria de la Informática.
https://upcommons.upc.edu/bitstream/handle/2099/11840/r51.pdf

Rodríguez del Pino, J. C., Rubio Royo, E., y Hernández Figueroa, Z. (2012).
A virtual programming lab for Moodle with automatic assessment and anti-plagiarism features.
En The Internacional Conference on *e*-Learning, *e*-Business, Entreprise Information Systems, & *e*-Government.
http://worldcomp-proceedings.com/proc/p2012/EEE3753.pdf