Diseño de una metodología docente para dos asignaturas de programación basada exclusivamente en la evaluación automática de programas

- **COORDINADOR**: Adolfo Rodríguez de Soto
- **ÁREAS DE CONOCIMIENTO**: Ciencias de la Computación e I.A. / Lenguajes y Sistemas Informáticos.
- **TITULACIÓN**: Grado en Ingeniería Informática
- **ASIGNATURAS IMPLICADAS**: Programación I. / Estructuras de Datos.

PAID 2015 - Universidad de León

CONTEXTO

- **Asignaturas de programación**: El aprendizaje de la programación tiene un alto nivel de dificultad.
  - "A programar se aprende programando"
  - Corrección manual muy laboriosa (sin ayudantes)
    - Nº alumnos: Entre 110 y 150 por asignatura
      - 5 ejercicios * 110 alumnos * 10 minutos/ejercicio=5500 minutos= 91 horas
- **OBJETIVO**: Introducir la **evaluación automática** de programas para **fomentar la participación del estudiante** en el aprendizaje.

PROGRAMACIÓN I

- Plugin de Moodle Virtual Programming Laboratory (VPL) activado en agora.unileon.es

- **Integración total con agora**. Calificaciones automáticamente incluidas en el libro de calificaciones.

- Permite evaluación automática, control de plagio, compilación, ejecución y depuración online. Editor integrado con control de corta/pega.

- Personalización a través del desarrollo de scripts. Ejecución controlada en máquina jaula.

PROGRAMACIÓN I

Metodología

- **Competencia principal de la asignatura:** saber hacer programas pequeños.
- La asignatura se desarrolla alrededor de los **ejercicios y prácticas de programación** a realizar durante el curso.
- De 10 a 12 ejercicios de programación durante el curso. Nuevos cada curso. Mezcla de ejercicios: a diseñar completamente, a completar diseño, a diseñar tests, etc.
- 2 prácticas de programación más grandes. Nuevas cada curso. Desarrollar completamente por el alumno.
- **Contenidos sincronizados** entre clases teóricas y los requerimientos de cada uno de los ejercicios semanales. Ejemplos orientados a solucionar los ejercicios.

Programación I

Evaluación

- Los ejercicios y prácticas de programación se evalúan mediante **tests automáticos**: público uno, ocultos cuatro o cinco. Evaluación continua para ambos. Los ejercicios se resuelven por el profesor una vez finalizado el plazo de entrega.

- **Control de plagio** en los ejercicios un poco más grandes y en las prácticas de programación. Forzar realización individual.

- **Puntuación del examen final**: sobre el propio trabajo realizado por el alumno durante el curso. Se permite perfeccionar la entrega durante el examen (editando el código entregado) y se pide programar una modificación, que se evalúa automáticamente.

25 de enero - 31 de enero

Lunes 26 de enero. Examen teoría y desempeño prácticas de programación.

- Solicitud examen Primera Convocatoria — Editar
- Asignación DEFINITIVA sesión examen Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión I. Ordenadores Pares. Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión I. Ordenadores Impares. Primera Convocatoria — Editar
- Práctica I. Modificación de examen 1. Primera Convocatoria. — Editar
- Práctica II. Modificación de examen 1. Primera Convocatoria. — Editar
- Ejercicio extra programación. Sesión I. — Editar
- Teoría. Examen Test. Sesión II. Ordenadores Impares. Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión II. Ordenadores Pares. Primera Convocatoria — Editar
- Práctica I. Modificación de examen 2. Primera Convocatoria. — Editar
- Práctica II. Modificación de examen 2. Primera Convocatoria. — Editar
- Ejercicio extra programación. Sesión II. — Editar
- Teoría. Examen Test. Sesión III. Ordenadores Impares. Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión III. Pablo González Álvarez. Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión III. Ordenadores Pares. Primera Convocatoria — Editar
- Práctica I. Modificación de examen 3. Primera Convocatoria. — Editar
- Práctica II. Modificación de examen 3. Primera Convocatoria. — Editar
- Ejercicio extra programación. Sesión III. — Editar
- Teoría. Examen Test. Sesión IV. Ordenadores Impares. Primera Convocatoria — Editar
- Teoría. Examen Test. Sesión IV. Ordenadores Pares. Primera Convocatoria — Editar
- Práctica I. Modificación de examen 4. Primera Convocatoria. — Editar
- Práctica II. Modificación de examen 4. Primera Convocatoria. — Editar
- Ejercicio extra programación. Sesión IV. — Editar

Calificación

Evaluada el miércoles, 27 de enero de 2016, 10:12 por Calificación automática
Calificación 40 / 100
Comentarios del revisor
[+]**Failed tests**
[+]**Test 1: 2_testDeRemplazo**
[+]**Test 2: 3_test1**
[+]**Test 5: 6_testLargo**
[+]**Summary of tests**
Entregada el lunes, 25 de enero de 2016, 11:12 (Descargar)

Evaluación automática[-]

Nota propuesta: 40 / 100
Compilación
Note: soplandoBolas.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Comentarios
[+]**Failed tests**
[+]**Test 1: 2_testDeRemplazo**
[+]**Test 2: 3_test1**
[+]**Test 5: 6_testLargo**
[+]**Summary of tests**

Calificación automática

Ejemplo Exámenes

Similaridad | **Lista de similitudes encontradas** | Lista de marcas de agua | Descargar las entregas

#	Nombre / Apellido(s)		Similar a	Cluster #
1	Bola.java 100 / 100 [redacted]	100|100|89***	Bola.java 80 / 100 [redacted]	1
2	Posicion.java 40 / 100 [redacted]	71|80|64**	Bola.java 60 / 100 [redacted]	1
3	Main.java 40 / 100 [redacted]	|93|72**	Main.java 40 / 100 [redacted]	3
4	ObjetivePosition.java 100 / 100 [redacted]	66||81**	Coordinate.java 60 / 100 [redacted]	1
5	Posicion.java 100 / 100 [redacted]	71|66|34*	Coordinate.java 60 / 100 [redacted]	1
6	Bola.java 100 / 100 [redacted]	66|92|66***	Cuadrito.java 40 / 100 [redacted]	1
7	Casilla.java 60 / 100 [redacted]	||78*	Bola.java 100 / 100 [redacted]	1
8	Objetivo.java 100 / 100 [redacted]	70|82|70**	Casilla.java 60 / 100 [redacted]	1
9	Bola.java 80 / 100 [redacted]	66|92|**	Cuadrito.java 40 / 100 [redacted]	1
10	Soluciones.java 40 / 100 [redacted]	55|76|73*	PosiblesSol.java 40 / 100 [redacted]	2

Ejemplo Informe Control de plagio

ESTRUCTURAS DE DATOS

Herramienta de Evaluación Automática elegida: Web-CAT

- En Web-CAT el proceso de evaluación se basa en **plugins**.
- En la asignatura de Estructuras de Datos se ha elegido el plugin de **desarrollo dirigido por las pruebas**, que consta de varios pasos:

1. **Compilación del código entregado por el alumno.** Detectan problemas como implementaciones incompletas de la solución, o diferencias entre el entorno de trabajo del alumno y del servidor, como diferentes codificaciones de los ficheros fuente, incompatibilidad de versiones de librerías como JUnit, o del propio JDK.
2. **Análisis estático del código fuente entregado:** Comprueba aspectos como convenios de nombres, complejidad de expresiones, uso de comentarios, etc.
3. **Ejecución de los tests entregados por el alumno y evaluación de la cobertura** que ofrecen.
4. **Ejecución de los tests de referencia preparados por el profesor,** que comprueban que la solución dada por el alumno es correcta.
5. **Generación y presentación al alumno de un informe preliminar con los resultados** de los pasos anteriores. Aquí **recibe advertencias** sobre errores detectados en el código para que pueda corregirlos y entregar la práctica de nuevo.

ESTRUCTURAS DE DATOS

Herramienta de Evaluación Automática elegida: Web-CAT

- El alumno **detecta sus propios errores y aprende de ellos.**

- Se permiten realizar varias entregas por el mismo alumno en la misma práctica.

  - Esto **motiva** al alumno a sumergirse en un ciclo de mejora continua, lo que repercute en mejores implementaciones, aumentando la calidad del código entregado y por tanto en puntuaciones más altas.

- El profesor dispone de **información completa de las entregas** de los alumnos, con acceso inmediato al **código** entregado y a los **resultados** obtenidos en cada entrega.

ESTRUCTURAS DE DATOS

Mensajes Mostrados en la Evaluación

Estimate of Problem Coverage (73%)

Problem coverage: 73%

For this assignment, the proportion of the problem that is covered by your test cases is being assessed by running a suite of reference tests against your solution, and comparing the results of the reference tests against the results produced by your tests.

Differences in test results indicate that your code still contains bugs. Your code appears to cover only 73% of the behavior required for this assignment.

Your test cases are not detecting these defects, so your testing is incomplete--covering at most only 73% of the required behavior, possibly even less.

Double check that you have carefully followed all initial conditions requested in the assign
of your program.

The following hint(s) may help you locate some ways in which your solution and your testin

- Timeout occurred. Please note the time in the report does not reflect the time unti
- Comprueba la uniÃ³n de bolsas vacÃas union of empty bags
- Comprueba la uniÃ³n con una bolsa vacÃa union with empty bag

(only 3 of 12 hints shown)

Advertencias de errores detectados

The following hint(s) may help you locate some ways in which your solution and your testing may be improved:

- Timeout occurred. Please note the time in the report does not reflect the time until the timeout.
- Comprueba la uniÃ³n de bolsas vacÃas union of empty bags
- Comprueba la uniÃ³n con una bolsa vacÃa union with empty bag

Code Coverage from Your Tests (90%)

Code Coverage: 90% (percentage of methods exercised by your tests)

You can improve your testing by looking for any lines highlighted in this color in your code listings above. Such lines have not been sufficiently tested--hover your mouse over them to find out why.

Results from Running Your Tests (94%)

The results of running your own test cases are shown below. Click on a failed test to see the reason for the failure and an execution trace that shows where the error occurred.

Passes: 33/35 Errors: 2/35 Failures: 0/35

- ule.edi.bag.ArrayBagImplTests (0.08 s)
- ule.edi.bag.BagsTests (0.032 s)
  - testComplement (0.03 s)
  - testEqualityOfNullReferences (0.002 s)
- ule.edi.bag.LinkedBagImplTests (0.029 s)
- ule.edi.currency.EuroCurrencyTests (0.006 s)

94% 6%

java.lang.NoSuchMethodError: org.hamcrest.Matcher.describeMismatch(Ljava/lang/Object;Lo
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:18)
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:8)
at ule.edi.bag.BagsTests.testComplement(BagsTests.java:53)

Output from Your Tests

VENTAJAS

- Posibilidad de **aumentar el número de ejercicios/prácticas** de programación que los alumnos realizan a lo largo del curso y evaluarlos en función de la eficacia del programa realizado.

- Orientación de las asignaturas a una **evaluación continua** y por **competencias**.

- **Aumento de la motivación** del alumno que recibe información inmediata de la calidad de su implementación. Se hace consciente de su propio aprendizaje.

- Se **evalúa y valora el esfuerzo realizado por el alumno** en el proceso de aprendizaje y no solamente el resultado final en los exámenes.

CONCLUSIONES

- La **evaluación automática** de los programas desarrollados por los alumnos en las asignaturas de programación constituye una **metodología muy adecuada** para lograr el objetivo de **adquirir las competencias** de capacidad para escribir programas y utilizar las estructuras de datos adecuadas.

- Además se añade la **adquisición de la competencia de capacidad para diseñar y construir casos de prueba** para sus implementaciones que garanticen la total cobertura del código y posibiliten la consecución de las especificaciones planteadas en los ejercicios.

- **Mejora el aprendizaje**, ya que permite que el alumno realice más ejercicios completos de programación a lo largo de la asignatura, obligándole a programar, y por tanto a aprender.

- Metodologías **aplicables a muchas asignaturas** del Grado en Ingeniería Informática, así como a cualquier asignatura de otros grados que implique aprender programación.