

---

# MONITORIZACIÓN DE PRUEBAS

*Título proyecto: Servidor de Contenidos*

*Ref. proyecto: equipo-10*

---

Validación e Verificación de Software

**17 de diciembre de 2015**

Esteban Abanqueiro Agrelo  
Manoel Antón Folgueira Hernández  
Samuel Paredes Sánchez  
Diego Sánchez Lamas

Fecha de aprobación	Control de versiones	Observaciones
17/12/2015	1.0	Ninguna

<https://github.com/galu8/ValidacionAutomatica>

# Índice

<b>1. Contexto</b>	<b>3</b>
<b>2. Estado actual</b>	<b>3</b>
<b>3. Registro de pruebas</b>	<b>6</b>
3.1. Cobertura . . . . .	6
3.2. Mockito . . . . .	7
3.3. CheckStyle . . . . .	8
3.4. GraphWalker . . . . .	8
3.5. QuickCheck . . . . .	9
3.6. JETM . . . . .	9
3.7. PMD . . . . .	10
3.8. FindBugs . . . . .	10
<b>4. Registro de errores</b>	<b>11</b>
<b>5. Estadísticas</b>	<b>12</b>
<b>6. Otros aspectos de interés</b>	<b>13</b>

## 1. Contexto

En este documento abarcaremos el proceso desde principio a fin de las pruebas realizadas en el proyecto durante las semanas 2 a 4. El plan de pruebas utilizado ha sido poco estructural, ya que no hemos separado cada caso de prueba en un método aparte, cosas que podría dificultar a terceros la comprensión de los mismos.

## 2. Estado actual

Las personas responsables del desarrollo de las funcionalidades y de su proceso de prueba son:

- Esteban Abanqueiro Agrelo
- Manoel Antón Folgueira Hernández
- Samuel Paredes Sánchez
- Diego Sánchez Lamas

Funcionalidades actuales:

- Módulo servidor:
  - obtenerNombre(): Cadena  
Devuelve el nombre del servidor.
    - Pruebas objetivo: 0
    - Pruebas preparadas: 1
    - Porcentaje ejecutadas: 100
    - Porcentaje superadas: 100
  - alta(): Cadena  
Se da de alta en el servidor un token normal aleatorio y lo devuelve.
    - Pruebas objetivo: 2
    - Pruebas preparadas: 2
    - Porcentaje ejecutadas: 100
    - Porcentaje superadas: 100
  - baja(token: Cadena)  
Se da de baja en el servidor el token normal especificado.
    - Pruebas objetivo: 3

- Pruebas preparadas: 3
- Porcentaje ejecutadas: 100
- Porcentaje superadas: 100
  
- agregar(contenido: Contenido, token: Cadena)
 

Se agrega el contenido en el servidor usando el token. El token debe ser especial, si no devuelve una excepción.

  - Pruebas objetivo: 2
  - Pruebas preparadas: 2
  - Porcentaje ejecutadas: 100
  - Porcentaje superadas: 100
  
- eliminar(contenido: Contenido, token: Cadena)
 

Se elimina el contenido en el servidor usando el token. El token debe ser especial, si no devuelve una excepción.

  - Pruebas objetivo: 4
  - Pruebas preparadas: 3
  - Porcentaje ejecutadas: 75
  - Porcentaje superadas: 75
  
- buscar(subcadena: Cadena, token: Cadena): Contenido [ ]
 

Se buscan en el servidor todos los contenidos que contienen la subcadena. Se devuelven hasta 10 contenidos si el token es normal, e ilimitados contenidos si el token es especial. Si no se especifica token se devuelven contenidos con publicidad.

  - Pruebas objetivo: 3
  - Pruebas preparadas: 3
  - Porcentaje ejecutadas: 100- Porcentaje superadas: 100
  
- Módulo contenido:
  - obtenerTitulo(): Cadena
 

Devuelve el título del contenido.

    - Pruebas objetivo: 0
    - Pruebas preparadas: 0
    - Porcentaje ejecutadas: 100
    - Porcentaje superadas: 100
  
  - obtenerDuracion(): Entero
 

Devuelve la duración del contenido en segundos.

- Pruebas objetivo: 5
  - Pruebas preparadas: 5
  - Porcentaje ejecutadas: 100
  - Porcentaje superadas: 100
- obtenerListaReproduccion(): Contenido [ ]  
 En el caso de emisoras devuelve su lista de reproducción, en el caso de otros tipos de contenido se devuelve a si mismo en una lista de un elemento.
    - Pruebas objetivo: 2
    - Pruebas preparadas: 1
    - Porcentaje ejecutadas: 50
    - Porcentaje superadas: 50
- buscar(subcadena: Cadena): Contenido [ ]  
 En el caso de emisoras busca la subcadena entre los títulos del contenido que hay en su lista de reproducción y recursivamente en sus emisoras internas. En el caso de otros tipos de contenido, se devuelven a si mismos en una lista de un elemento si la subcadena está contenida en su título.
    - Pruebas objetivo: 5
    - Pruebas preparadas: 5
    - Porcentaje ejecutadas: 100
    - Porcentaje superadas: 100
- agregar(contenido: Contenido, predecesor: Contenido): Cadena  
 En el caso de emisoras agrega el contenido en la lista de reproducción a continuación del predecesor especificado. Si el predecesor es nulo, se agrega al final. En el caso de otros tipos de contenido no tiene efecto.
    - Pruebas objetivo: 6
    - Pruebas preparadas: 6
    - Porcentaje ejecutadas: 100
    - Porcentaje superadas: 100
- eliminar(contenido: Contenido)  
 En el caso de emisoras elimina (si lo tiene) el contenido de su lista de reproducción. En el caso de otros tipos de contenido no tiene efecto.

- Pruebas objetivo: 4
- Pruebas preparadas: 4
- Porcentaje ejecutadas: 100
- Porcentaje superadas: 100

### 3. Registro de pruebas

#### 3.1. Cobertura

Para comprobar hasta que punto los test realizados comprobaban la validez del código implementado, hemos usado la herramienta cobertura. En el estado inicial antes de realizar la completitud de la misma, nuestros test abarcaban alrededor del 70-75 por ciento, dejando sólo la pequeña parte de métodos triviales como gets, etc.

Sin embargo, hemos considerado que aunque los métodos sean sencillos, es buena práctica realizar el mayor número de comprobaciones de cada línea de nuestro código, por lo tanto hemos completado la cobertura llegando hasta un 98 por ciento de cobertura de línea y un 95 por ciento de cobertura de ramas:

Packages		Coverage Report - All Packages				
		Package	# Classes	Line Coverage	Branch Coverage	Complexity
All		All Packages	12	98%	95%	1,967
es.udc.vvs.va.model.contenido		es.udc.vvs.va.model.contenido	5	100%	100%	2,231
es.udc.vvs.va.model.exceptions		es.udc.vvs.va.model.exceptions	1	100%	N/A	1
es.udc.vvs.va.model.servidor		es.udc.vvs.va.model.servidor	4	97%	90%	1,963
es.udc.vvs.va.test.model.testautomation		es.udc.vvs.va.test.model.testautomation	1	N/A	N/A	1
es.udc.vvs.va.util		es.udc.vvs.va.util	1	0%	N/A	0
es.udc.vvs.va.model.servidor		Report generated by Cobertura 2.1.1 on 17/12/15 11:22.				
Classes						
Servidor (N/A)						
ServidorAbstracto (96%)						
ServidorPlano (100%)						
ServidorTransitivo (100%)						

Por lo tanto, podemos decir que abarcamos prácticamente toda la implementación de nuestro proyecto. Sin embargo esto no quiere decir que la funcionalidad de nuestra aplicación compruebe al 100 por ciento el correcto funcionamiento de la aplicación.

Se han añadido test como agregar una emisora en otra (éxito), búsqueda recursiva y crear un servidor transitivo sin servidor relacionado.

#### 3.2. Mockito

Con el fin de completar las pruebas de unidad (recordemos que las Pruebas de unidad son del tipo que determinan si un componente funciona de cierta forma esperada, bajos ciertos escenarios controlados y bien definidos, y excluyen el buen o mal funcionamiento de los elementos que colaboran con el mismo,

pues no es de la incumbencia del elemento bajo ejecución) debemos usar para la implementación del servidor mocks de Contenidos.

Para el caso de Emisora, dicha clase llama a métodos de Anuncio y Cancion, por lo tanto consideramos que los métodos de la clase emisora que llaman a otros métodos de esta misma clase debemos probarlos con mocks:

```
c1 = mock(Cancion.class);
c2 = mock(Cancion.class);
c3 = mock(Cancion.class);
a = mock(Anuncio.class);

when(c1.obtenerDuracion()).thenReturn(6);
when(c2.obtenerDuracion()).thenReturn(4);
when(c3.obtenerDuracion()).thenReturn(5);
when(a.obtenerDuracion()).thenReturn(5);

when(c1.obtenerTitulo()).thenReturn("Cancion 1");
when(c2.obtenerTitulo()).thenReturn("Cancion 2");
when(c3.obtenerTitulo()).thenReturn("Cancion 3");
when(a.obtenerTitulo()).thenReturn("PUBLICIDAD");

e = new Emisora("Emisora 1");
e.agregar(c1, null);
e.agregar(c2, c1);
e.agregar(c3, c2);
e.agregar(a, c3);
```

En la imagen de arriba podemos ver un ejemplo de cómo se crean objetos mock y como definimos lo que devuelve cada método de los mock objects

Con ServidorPlano y transitivo hemos realizado algo similar a lo mostrado con Emisora.

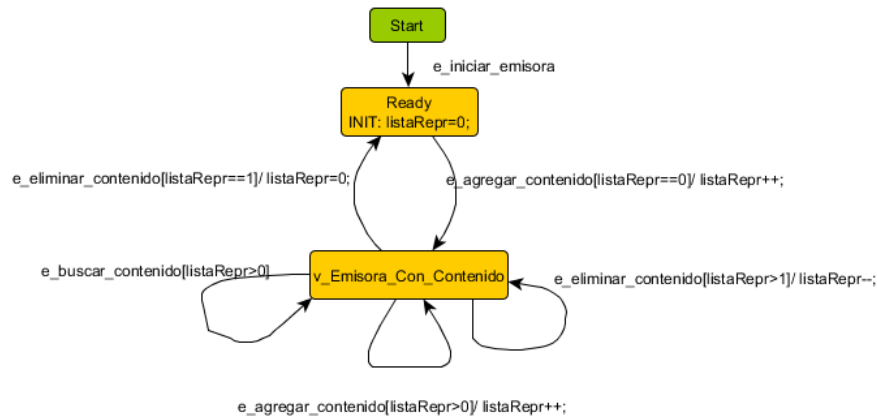
### 3.3. CheckStyle

Hemos usado checkstyle con el fin de analizar nuestro código y comprobar que sigue las métricas de calidad correspondientes.

Una vez instalado el plugin, hemos decidido usar la configuración Google check porque nos parece la más correcta debido a la excelente imagen que tiene la compañía.

### 3.4. GraphWalker

GraphWalker es una herramienta para automatizar la generación y ejecución de pruebas dinámicas. Partiendo de un diagrama de estados (que debe seguir unas reglas concretas) genera una interfaz. En nuestro caso hemos creado un diagrama de estados que representa la agregación de contenidos a una emisora:



Este grafo se introduce en `src/main/resources` y ejecutando el goal `mvn graphwalker:generateSources` nos crea una interfaz en `target` dentro de la misma estructura de directorios que la que hayamos incluido.

Dicha interfaz debe implementarse, generando para cada estado y cada transición las comprobaciones correspondientes.

Principalmente se ejecutan 3 tests:

- Smoke test  
Verifica el flujo básico del modelo. Usando el algoritmo A \*, crea un camino recto desde el punto de partida , `e_Init`, en el gráfico, al vértice `v_Browse`.
- Functional test example  
Cubre el gráfico completo. Se inicia a partir `e_Init`, y termina tan pronto como la condición de parada se ha cumplido. Le indicamos que pase al menos una vez por cada transición.
- Stability test example  
Pedimos a GraphWalker que recorra el modelo al azar, hasta que se cumpla la condición de parada. Eso sucederá cuando haya pasado el



tiempo definido. En una prueba real, podrían ser de 30 minutos, o incluso horas.

### 3.5. QuickCheck

Creamos un generador de canciones, un generador de contenidos (canciones o emisoras aleatoriamente) y un generador de duraciones (enteros positivos). Estos generadores los utilizamos para comprobar el correcto funcionamiento de `obtenerDuracion()` tanto en emisoras como en canciones.

### 3.6. JETM

Aprovechando los generadores de QuickCheck, realizamos pruebas de rendimiento de diferentes operaciones. Las funcionalidades que probamos son:

- buscar en Canción
- obtenerDuracion en Canción
- buscar en una Emisora con solo canciones
- buscar en una Emisora con contenidos
- obtenerDuracion en una Emisora con solo canciones
- obtenerDuracion en una Emisora con contenidos
- buscar en un ServidorPlano con solo canciones
- buscar en un ServidorPlano con contenidos
- buscar en un ServidorTransitivo con solo canciones (que están aleatoriamente en el servidor de respaldo o en el transitivo)
- buscar en un ServidorTransitivo con contenidos (que están aleatoriamente en el servidor de respaldo o en el transitivo)

Estos fueron los resultados obtenidos en una de las ejecuciones que realizamos:

[INFO ] [EtmMonitor] JETM 1.2.3 started.

Measurement Point	#	Average	Min	Max	Total
Cancion:buscar	2000	0,001	0,000	0,037	1,131
Cancion:obtenerDuracion	2000	0,000	0,000	0,012	0,483
Emisora:buscarCanciones	2000	0,002	0,000	0,054	3,852
Emisora:buscarContenidos	2000	0,001	0,000	0,033	1,207
Emisora:obtenerDuracionCanciones	2000	0,001	0,000	0,038	1,170
Emisora:obtenerDuracionContenidos	2000	0,000	0,000	0,015	0,651
ServidorPlano:buscarCanciones	2000	0,019	0,000	33,713	37,275
ServidorPlano:buscarContenidos	2000	0,001	0,000	0,013	1,355
ServidorTransitivo:buscarCanciones	2000	0,005	0,002	0,538	10,590
ServidorTransitivo:buscarContenidos	2000	0,001	0,000	0,039	1,447

[INFO ] [EtmMonitor] Shutting down JETM.

Como vemos en la imagen, realizamos 2000 ejecuciones de cada una de las funcionalidades testeadas.

### 3.7. PMD

PMD es un analizador de código estático. Usa un serie de “rule-sets” que definen cuándo es incorrecto un fragmento de código. También permite incluir nuevas reglas que podremos definir y podemos configurar las reglas que se ejecutan.

En nuestro caso incluimos todas las disponibles y encontró varios errores que documentamos en la siguiente sección.

### 3.8. FindBugs

FindBugs encuentra errores potenciales, en nuestro caso no encontró ninguno:

## Validacion Project

Project Documentation


Project Information

Project Reports

Code Coverage

FindBugs

PMD

Built by 

Last Published: 2015-12-17

Version: 0.0.1-SNAPSHOT

### FindBugs Bug Detector Report

The following document contains the results of [FindBugs](#)

FindBugs Version is 3.0.1

Threshold is *medium*

Effort is *min*

### Summary

Classes	Bugs	Errors	Missing Classes
12	0	0	0

### Files

Class	Bugs
-------	------

## 4. Registro de errores

Errores encontrados durante las pruebas:

- Mockito: Durante la prueba de MockEmisoraTest, hemos encontrado un error en la búsqueda de emisora en la cual no comprobábamos que la subcadena no distinguiese entre mayúsculas y minúsculas. Esto nos ha pasado debido a que desde el servidor sí se hace y en las pruebas iniciales dicho error se nos ha escapado, por lo tanto nos ha servido de utilidad el uso de mocks para encontrar estos pequeños errores puntuales que, aunque en un proyecto pequeño puede ser sencillo de encontrar, a la hora de realizar un proyecto más grande, podría generar muchos problemas el encontrar un error tan puntual.
- CheckStyle:
  - Uso de tabuladores en vez de espacios.
  - Sangrías incorrectas.
  - Orden incorrecto en la organización de los imports.
  - No uso de llaves para sentencias y bucles de una línea.
  - Javadoc mal generado.
  - Nombres de métodos con más de una mayúscula.
  - Uso del mismo nombre para variables locales.

```

5 import es.udc.vvs.va.model.contenido.Contenido;
6 import es.udc.vvs.va.model.exceptions.ContentManagerException;
7
31 */
32 public ServidorTransitivo(String nombreServidor, String token,
33     Servidor servidorRelacionado)
34     throws ContentManagerException {
35     super(nombreServidor, token);
36
37     if (servidorRelacionado == null) {
38         throw new ContentManagerException("Los servidores transitivos "
39             + "necesitan un servidor de apoyo.");
40     }
41
42     this.servidorRelacionado = servidorRelacionado;
43 }

```

Para agilizar el proceso de corrección, hemos configurado en eclipse el Java Code style de nuestro proyecto con una configuración más acorde con las convenciones de Google checks, y de esta forma ir dando formato a cada clase mucho más rápido. También hemos usado el plugin JautoDoc para la generación correcta de javadoc en los sitios en los que faltaba.

- PMD:
  - Las clases abstractas deben llamarse «AbstractXXX».
  - Este punto no lo consideramos un error ya que estamos programando en español.
  - Evitar sentencias condicionales sin llaves.
  - Evitar variables con nombres cortos como `i`.
  - Cada test de JUnit debería contener tan solo un assert.
  - Assert booleano innecesario.
  - Evitar `assertTrue` o `assertFalse` con un parámetro `"x==y"`, usar en su lugar `assertSame(x,y)` o `assertNotSame(x, y)`.

## 5. Estadísticas

- Número de errores encontrados semanalmente: 1
- Nivel de progreso en la ejecución de las pruebas: alto
- Perfil de detección de errores:
- Errores críticos abiertos: 5

- Errores críticos cerrados: 5
- Calidad actual: alta
- Estabilidad actual: alta

## **6. Otros aspectos de interés**

Nada a destacar.