MONITORIZACIÓN DE PRUEBAS

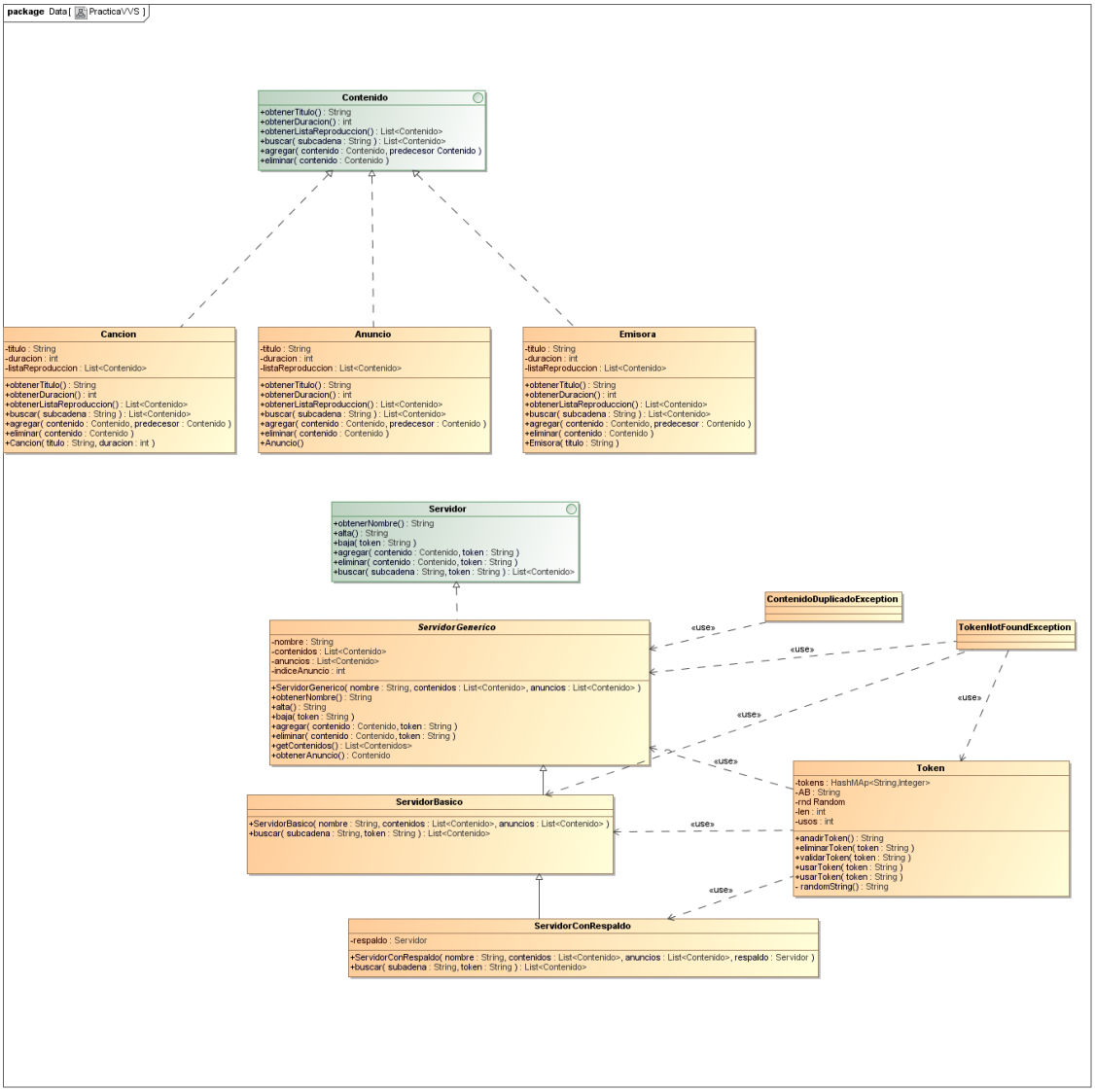
Título proxecto: PracticaVVS
Ref. proxecto: https://github.com/angelcastro2/PracticaVVS

Validación y Verificación de Software

Fecha de aprobación	Control de versiones	Observaciones
16/12/2015	1.0	Pedro Fernández, Ángel Castro, Avelino Ríos

1. Contexto

Antes de realizar las pruebas, cuando terminamos de codificar la lógica de la práctica teníamos el siguiente diagrama de clases:



Éste diagrama tiene implementadas las funcionalidades requeridas por el software para esta práctica. Partiendo del enunciado propuesto nos encontramos con las siguientes funcionalidades:

CONTENIDO

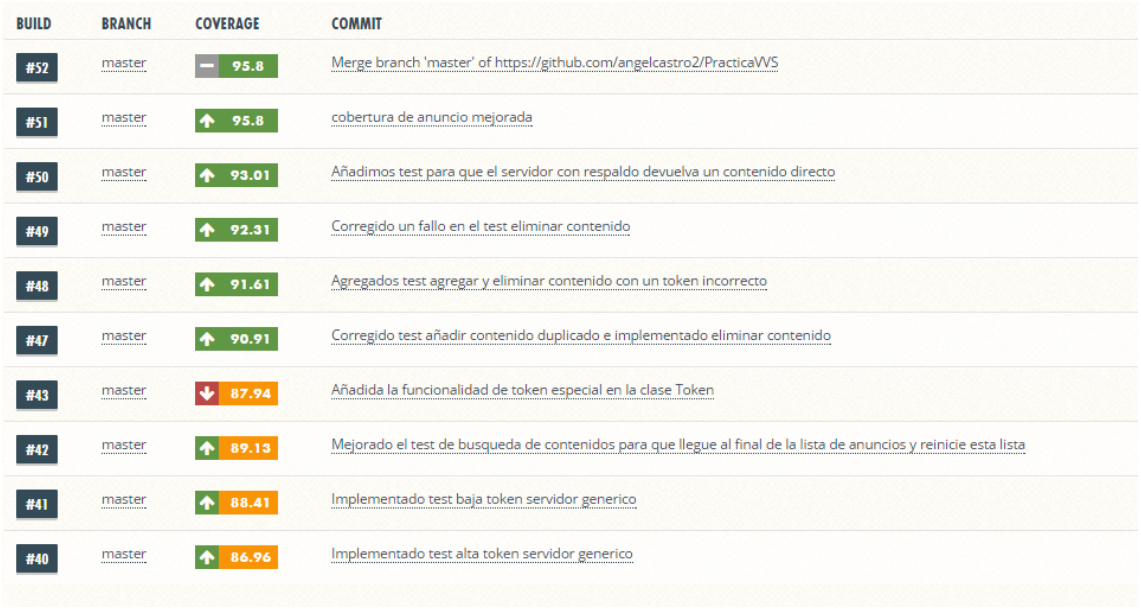
- Obtener titulo.
- Obtener duración.
- Obtener lista de reproducción.
- Buscar utilizando una subcadena.
- Agregar y eliminar no tendrán efecto en canción y anuncio, pero sí en emisora.

SERVIDOR

- Obtener nombre del servidor.
- Dar de alta un token.
- Dar de baja un token que aún no está agotado.
- Agregar contenidos utilizando un token especial.
- Eliminar contenidos utilizando un token especial.
- Buscar contenidos utilizando un token normal.

2. Estado actual

A partir de lo descrito en el apartado interior la primera modalidad de testing que realizamos fue happy testing. Utilizamos para ello JUnit intendando cubrir la mayor cantidad de código posible. A continuación se puede ver una captura de la cobertura conseguida.



BUILD	BRANCH	COVERAGE	COMMIT
#52	master	95.8	Merge branch 'master' of https://github.com/angelcastro2/PracticaVVS
#51	master	95.8	cobertura de anuncio mejorada
#50	master	93.01	Añadimos test para que el servidor con respaldo devuelva un contenido directo
#49	master	92.31	Corregido un fallo en el test eliminar contenido
#48	master	91.61	Agregados test agregar y eliminar contenido con un token incorrecto
#47	master	90.91	Corregido test añadir contenido duplicado e implementado eliminar contenido
#43	master	87.94	Añadida la funcionalidad de token especial en la clase Token
#42	master	89.13	Mejorado el test de busqueda de contenidos para que llegue al final de la lista de anuncios y reinicie esta lista
#41	master	88.41	Implementado test baja token servidor generico
#40	master	86.96	Implementado test alta token servidor generico

Posteriormente realizamos las pruebas descritas en el siguiente apartado.
Las pruebas han sido realizadas por los tres miembros del grupo: Ángel Castro, Pedro Fernández y Avelino Ríos.
Tras realizar todos los test, creemos que podemos tener un nivel alto de confianza en que nuestro código está correcto puesto que ha sido verificado por todos los costados con las herramientas propuestas que creímos necesarias. Para la elección de las herramientas nos hemos basado en el objetivo de cada una de ellas de forma que hemos elegido herramientas para test de unidad, integración, mutación y generación de datos dinámica durante la ejecución de las pruebas, además de emplear otras herramientas para realizar pruebas estructurales o estáticas.

3. Registro de pruebas

Las pruebas realizadas por el grupo hasta la fecha de entrega han sido las siguientes:

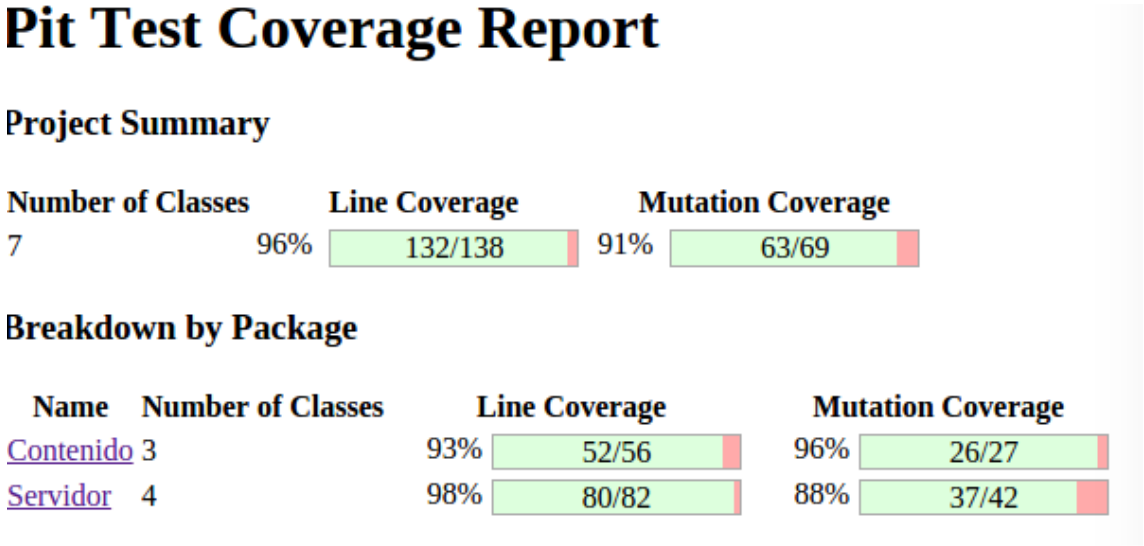
- Pruebas de unidad en Contenido con JUnit.
- Pruebas de integración con JUnit tanto a Servidor Básico como a Servidor con respaldo.
- Uso de la herramienta Coveralls para mejorar la cobertura de nuestros test.

- Herramienta Quickcheck para comprobar que las funciones cumplen con las propiedades que se esperan de ellas.
- Hemos utilizado Mockito para realizar test unitarios en la parte del servidor.
- Graphwalker para generar test automaticos, hemos realizado el diagrama conveniente con yeD.
- Para mejorar la calidad de nuestros test hemos realizado pruebas de mutación con Pit testing.
- Hicimos uso de la herramienta FindBugs para la búsqueda de variables muertas, llamadas ineficientes o para comporbar que los modificadores de las variables sean correctos.
- Comprobamos estilo del código sea correcto gracias a CheckStyle.
- Prácticas cuestionables a la hora de codificar por medio de PMD.
- Finalmente, destacar que hemos utilizado integración contínua por medio de Travis CI.

4. Registro de errores

Algunos apuntes relevantes a comentar sobre las pruebas realizadas, empezaremos con Pit Testing donde hemos podido corregir algunos errores como:

- El test eliminar contenido de emisora no comprobamos que la nueva duración se decrementa.
- El test buscar contenido en servidor, no comprobamos el orden en el que se insertan los anuncios.
- Varios errores en condiciones if y for (Ej: El test era correcto tanto si una condición de un if era $(a > 0)$ como $(a >= 0)$.
- Como el token es aleatorio, no podemos forzar un test que pase por ahí por eso es un caso no cubierto.
- No se han implementado los métodos agregar y eliminar de anuncio y canción, ya que no son necesarios, se han agregado los métodos vacíos para que cumplieran con la interfaz de contenido. Por este motivo, esas lineas no están cubiertas por los tests.



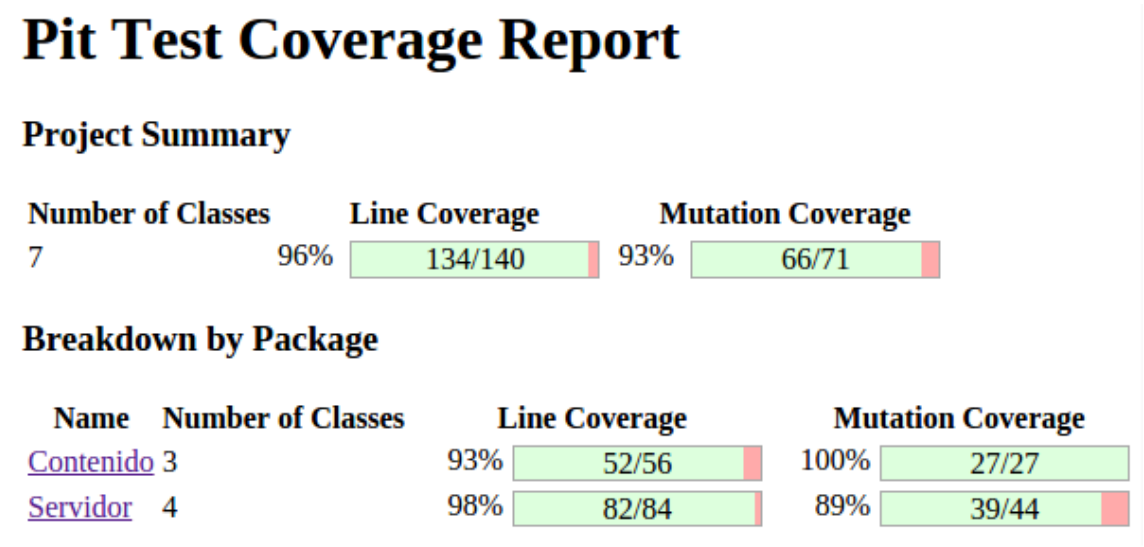
Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
7	96%	132/138	91%	63/69

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
Contenido	3	93%	52/56	96%	26/27
Servidor	4	98%	80/82	88%	37/42



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
7	96%	134/140	93%	66/71

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
Contenido	3	93%	52/56	100%	27/27
Servidor	4	98%	82/84	89%	39/44

Si nos fijamos en las gráficas anteriores se puede comprobar que tras utilizar dicha herramienta hemos obtenido una ligera mejora.

Con Mockito hemos encontrado lo siguiente:

- Fallo en el constructor de servidor porque no se le podían pasar nulos.

Con CheckStyle, FindBugs y PMD la mayoría de errores obtenidos han sido en referencia a:

- Valores asignados a variables locales que no se están utilizando.
- Atributos que deberían ser final.
- Métodos que no cumplen convenciones de nombrado.
- El valor de retorno de la función es ignorado.
- Algunos objetos que nunca se escriben.
- Nombre incorrecto de la clase abstracta, debería comenzar AbstractXXX.
- Un campo tiene el mismo nombre que un método(token).
- Evitar usar literales en los condicionales.
- Paquete que no cumple las notaciones de nombrado porque empieza con mayúscula.
- Falta de comentarios en constructor vacío.

Con QuickCheck no hemos encontrado ningún error.

5. Estadísticas

Número de errores semanales:

- Quickcheck: 0
- GraphWalker: 1
- Mockito: 1
- Coveralls: 3
- PIT: 6
- JUnit: 4
- FindBugs: 20
- Checkstyle: 1930

- PMD: 264

Para éste nos hemos basado en las issues que hemos abierto durante ese período y también mirando los commits realizados.
Perfil de detección de errores:

- Quickcheck: pruebas dinámicas.
- GraphWalker: pruebas dinámicas.
- Mockito: dinámicas de unidad.
- Coveralls: ninguna.
- PIT: mutación.
- JUnit: integración y unidad.
- FindBugs, Checkstyle y PMD: estructural/estático.

Como ya hemos comentado en apartados anteriores, en cuánto a la evaluación global del estado de calidad y estabilidad actuales podemos afirmar tras realizar todas estas pruebas, que tenemos un nivel bastante alto de confianza en nuestro sistema. Esta confianza es debida a que la cobertura de nuestro código es bastante elevada y hemos testeado la práctica desde los diferentes flancos propuestos y de los que consideramos oportunos y necesarios.

6. Otros aspectos de interés

Creemos que es interesante destacar que CheckStyle, PMD y FindBugs a veces dan algunos errores poco afortunados y falsos positivos, ademas de crear confusión en ciertos aspectos a la hora de testear la práctica. Un error bastante curioso, encontrado en la herramienta de integración contínua Travis CI es que cuando compilamos y ejecutamos test, si el archivo de log pasa de un tamaño determinado, aunque los test y la práctica estén correctos va a devolver Build Failure o Build Error. Para solucionar éste problema lo que hicimos fue acortar el tiempo de test en GraphWalker de modo que generará un fichero de log significativamente más pequeño.