Práctica 3

Tecnologías para el Desarrollo de Software
Grado en Ingeniería Informática
Mención Ingeniería del Software

Y. Crespo

Curso 2019-2020

Tarea a realizar

En esta Práctica se implementará el diseño basado en TDD y *Test First* realizado en la Práctica 2.

En esta Práctica el objetivo es obtener los tests en VERDE. Es decir, realizar la fase Green del ciclo Red-Green-Refactor.

Adicionalmente se añadirá una nueva funcionalidad a la clase que implementa un boletín de noticias.

```
Desiderata:
Se quiere analizar dos boletines en cuanto a grado de similitud en sus noticias.
Esto nos indicará un porcentaje de cuántas noticias en el boletín de referencia
son similares a noticias del otro boletín.
```

En el control de versiones será importante apreciar cuándo, cómo y en qué condiciones se añade esta nueva funcionalidad.

En esta práctica se medirá el grado de cobertura alcanzado y se intentará maximizar dicho grado de cobertura con la menor cantidad de test posibles. También se tomará la medida de la métrica estática ratio *code to test* así como la complejidad ciclomática por método y por clase de las clases implementadas.

La implementación se realizará aplicando *pair programming*, integración continua y gestión automática de dependencias.

Control de versiones

Se llevará un control de versiones utilizando `git` y se utilizará `gitLab` (`gitlab.inf.uva.es`) como repositorio centralizado y `Atlassian Bitbucket` como repositorio de respaldo. El repositorio en `gitLab` deberá alojarse en un **grupo** creado para ello llamado practica3. El proyecto que se alojará en el grupo practica3 deberá obtenerse mediante un fork del proyecto realizado en la práctica 2.

Al finalizar, para realizar la entrega, se añadirá a la profesora al proyecto (cuando ya no se vayan a realizar más *commits* y *pushes* al repositorio centralizado en `gitLab` y al respaldo en `Atlassian Bitbucket`) tal y como se indica en las **normas de entrega** (ver página 5). En el repositorio no se tendrá ningún `.class` o `.jar` ni tampoco la documentación de las clases que puede generarse con *javadoc* en cualquier momento.

En esta práctica **SÍ** se valorará el uso de `git` en cuanto a ramificación (*branchs*) y fusión (*merge*). Las ramas del proyecto incluirán: la rama *master*, una rama *develop* y se tendrá una **rama por tarea (*issue*)**.

Se aplicarán técnicas de programación por pares (*pair programming*), por lo que cada tarea de programación estará asignada a la pareja pero deberá intercambiarse el rol de "conductor" (*driver*) y el de "copiloto" (*observer or navigator*). Se revisará el historial de *commits* realizados en cada rama y se valorará que se realicen *commits* por los diferentes miembros del equipo, lo cual indicará su rol de "conductor" en ese momento.

Cuando se finaliza una tarea, se deberá integrar en la rama *develop*. Cuando se realicen pruebas y depuración deberá siempre hacerse en una rama diferente a la rama *develop*, incluso diferente a la rama en la que se haya estado desarrollando la tarea de programación asignada, creando un issue para esto en el issue tracker.

Las integraciones en *master* deberán hacerse solamente de partes funcionales testadas, con los tests en verde.

Se realizará primero la programación de la clase boletín. No se esperará a tener todas las tareas de programación terminadas para integrar en máster. Para conseguir esto, se usarán las pruebas en aislamiento. El boletín se aisla de la implementación de la noticia, consiguiendo los tests en verde.

Integración Continua

El proyecto estará gestionado automáticamente mediante ant y maven.

En el repositorio `gitLab` no se tendrá ningún .class o .jar pero deberá bastar hacer un *pull* o *clone* del repositorio y utilizar el script ant (build.xml) para compilar, ejecutar, ejecutar los tests, analizar la cobertura, etc. Por ello, se gestionarán las dependencias externas del proyecto mediante maven. El script ant básicamente se utilizará para delegar en maven y así no tener que memorizar las tareas y parámetros maven necesarios para realizar el objetivo.

El proyecto deberá responder a un arquetipo maven descrito con el pom.xml correspondiente. Estará basado en el arquetipo maven-archetype-quickstart. La versión básica del pom.xml obtenida a partir del arquetipo indicado se debe revisar y modificar para añadir la gestión de dependencias, actualizar versiones así como la configuración de plugins necesarios para el análisis de cobertura, el análisis de calidad y su generación de informes y cualquier otro que los autores necesiten.

Como se ha indicado anteriormente, para comenzar el trabajo se realizará un *fork* del proyecto `gitLab` realizado para la práctica 2 para obtener el proyecto de la práctica 3. El proyecto de la práctica 2 que ha sido bifurcado se convertirá en un proyecto maven con el arquetipo indicado. Posteriormente se realizará un *commit & push* con la conversión del proyecto a maven.

En `http://uvadoc.uva.es/handle/10324/30846` se encuentra un vídeo tutorial en el que se realiza y se explica esta tarea paso a paso, si bien puede realizarse mucho más rápido en bloque.

El script ant (build.xml) deberá contar al menos con los siguientes objetivos que tendrán que llamarse exactamente así:

compilar garantiza la obtención de todas las dependencias y genera los .class a partir de los fuentes (depende de limpiar). Este objetivo es el default.

ejecutarTestsTDDyCajaNegra se ejecutan los tests categorizados como TDD y caja negra sin incluir secuencia.

ejecutarPruebasSecuencia se ejecutan solamente las pruebas de secuencia pruebas de secuencia.

ejecutaCajaBlanca se ejecutan solamente las pruebas categorizadas de caja blanca que se añadieron para aumentar la cobertura

ejecutarTodoSinAislamiento incluirá la ejecución de los tres objetivos anteriores.

ejecutarTodosEnAislamiento ejecuta todos los tests de boletín que se tengan basados en mocks.

obtenerInformeCobertura obtener los informes de cobertura de sentencia, de rama, decisión/condición obtenidos con el total de todos los tests realizados (sin incluir los tests en aislamiento).

documentar genera la documentación del proyecto, incluida la carpeta doc con la documentación autocontenida en los fuentes mediante javadoc. La carpeta doc no debe alojarse nunca en el control de versiones.

limpiar deja el proyecto limpio eliminando todo lo generado con las ejecuciones de los objetivos.

calidad realiza un análisis de la calidad del proyecto utilizando sonarqube en modo *publish*.

Como se ha mencionado antes, para la consecución de estos objetivos será recomendable descansar en llamadas a maven.

En gitLab se utilizará el mecanismo de integración continua basado en sintaxis yaml para lo que se podrá utilizar como base el archivo .gitlab-ci.yml que se provee en el ejemplo: `https://gitlab.inf.uva.es/ejemplos-tds/tdsCI/blob/master/.gitlab-ci.yml`.

Para usuarios del IDE Eclipse: se podrá usar el lint de sonarqube como plugin en Eclipse. En Eclipse ir al menú Help $->$ Eclipse Marketplace. Buscamos Sonarlint y damos a instalar. Una vez instalado, si dais click derecho en el proyecto en Eclipse veréis una opción de Menú llamada SonarLint. Hay que hacer un bind del proyecto. La configuración del servidor sonar que tenéis que usar es `https://sonarqube.inf.uva.es/`. En dicho servidor os valen las credenciales que tenéis para los labs de la Escuela, mismas que en gitlab.inf.uva.es. Sin embargo no es seguro introducir login y password en Eclipse. Es mejor usar la opción token. Se deberá generar un token en el apartado Security de la configuración de usuario en el servidor `https://sonarqube.inf.uva.es/`.

Se debe hacer un bind al proyecto con key tds-general-for-sonarlint preparado especialmente para ello. A partir de ese momento, podéis usar el lint, la primera vez dais la opción Analyze, las siguientes veces podéis dar la opción Analyze changed files. Esta información se la brinda git al lint, proporciona los cambios y entonces se realizan los análisis sólo sobre los cambios que se han hecho desde el análisis anterior.

En el archivo .yml que os doy como referencia para el pipeline de gitLab tenéis explicado cómo lanzar un análisis que quede registrado en el servidor. Haced esto de vez en cuando para ir registrando la evolución de la calidad de vuestro proyecto.

Tests

Los tests serán implementados con JUnit. Se utilizará jUnit 4 o jUnit 5 consecuentemente con lo realizado en la práctica 2.

Además de los tests realizados en la Práctica 2, se realizarán pruebas de caja blanca para aumentar la cobertura.

Se probará en aislamiento la clase que implementa un boletín (se aislará de la clase que implementa una noticia) utilizando mocks objects (basados en EasyMock o Mockito a elección del equipo).

Se espera que el resultado de la ejecución de los tests en esta práctica sea **Verde** en todos los casos, tanto en las pruebas en aislamiento basadas en mocks como en las pruebas con los objetos reales.

Los tests en aislamiento basados en mocks se categorizarán utilizando la anotación `@Category` para indicar su naturaleza con la categoría:
`@Category(Isolation.class)`.

Los test de caja blanca realizados para aumentar la cobertura se categorizarán como:
`@Category(WhiteBox.class)`.

Se debe tener en cuenta que se pueden aportar varias categorías a una clase de test o a un test individual. Por ejemplo:
`@Category({WhiteBox.class, Isolation.class})`.

En caso de JUnit 5 se utilizarán los tags en los mismos casos antes explicados pero usando `@Tag(‘‘Isolation’’)`, etc.

Al final del documento se pueden consultar los criterios de evaluación de esta práctica tal y como se muestra en la Figura 1 de la página 7.

Issues, estimación y tiempo empleado

El proyecto en `gitlab.inf.uva.es` tendrá un listado de *issues* que se irán creando bien al comenzar el proyecto, bien durante el desarrollo del mismo. Al crear cada *issue*, se asignará a un miembro del equipo, se estimará (comando corto `/estimate`) el tiempo que se cree que será necesario para completar el *issue* y al finalizar, se indicará el tiempo que realmente ha sido necesario (comando corto `/spend`). En esta práctica, como en la anterior, no se valorará cómo se ha dividido el proyecto en *issues*, ni tampoco tendrá ningún efecto en la nota la diferencia entre el tiempo estimado y el empleado. Lo que se valora es que se haya estimado y se haya registrado lo real.

Características del proyecto Maven

- En esta práctica no será necesario, si no se desea, utilizar Eclipse en la implementación.
- El proyecto será un proyecto maven por lo que será necesario un archivo pom.xml correspondiente al arquetipo maven-archetype-quickstart.
- La estructura de carpetas deberá corresponderse con la definida por el arquetipo maven antes indicado.
- Deberá ser un proyecto válido para jdk 8 o 9, a elección del equipo.
- En el archivo pom.xml los tags `groupId`, `artifactId` y `name` tendrán el siguiente contenido:

```
<groupId>2019-2020-tds-x</groupId>
<artifactId>p3</artifactId>
<name>2019-2020 práctica 3 de TDS del equipo x</name>
```

- En los datos anteriores la x deberá ser sustituida por el número del equipo de prácticas. Este número se puede conocer por el canal Rocket creado para el equipo en `rocket.inf.uva.es`.

- Las clases y sus tests estarán en el mismo paquete aunque en carpetas de fuentes diferentes (las carpetas que genera para ello el arquetipo maven).
- El proyecto maven deberá indicar el conjunto de caracteres utilizado en los fuentes para evitar problemas de importación en entornos diferentes.

Normas de entrega

- El repositorio `git` y el centralizado en `gitLab`, así como el respaldo en `Atlassian Bitbucket`, deberá llamarse equipox-2019. Sustituyendo la x como se ha indicado anteriormente.
- Cualquier *push* al repositorio una vez realizada la entrega será penalizado con 0 en la Práctica.
- La entrega se realizará añadiendo a la profesora (usuario `yania` en `gitLab` y `yania@infor.uva.es` en `Atlassian Bitbucket`) con rol Reporter al repositorio que contiene el proyecto en `gitLab` (y con permisos de sólo lectura al respaldo en `Atlassian Bitbucket`) cuando no se vaya a realizar ningún otro *commit & push*.
- El script ant (build.xml) debe tener los objetivos descritos en el enunciado.
- Al entregar la práctica todo deberá quedar integrado en la rama *master*.
- En el tracking de versiones y por tanto en el repositorio remoto solamente residirán los archivos de la configuración del proyecto en el IDE que se elija, los fuentes de las clases de la solución, los fuentes de los tests y los archivos pom.xml (maven), build.xml (ant) y gitlab-ci.yml (pipeline de `gitLab`).
- En los fuentes entregados se hará referencia a los autores de la práctica en cada archivo fuente con el tag `@author` del *javadoc* de la cabecera de la clase y solamente en ese punto.
- El proyecto tendrá un archivo `README.md` que indicará toda la información que quieran aportar los autores sobre su proyecto además de la siguiente información:
  - Tiempo total en horas-hombre empleado en la realización de la práctica.
  - Clases que forman parte de la solución y por cada clase: SLOC (Source Lines of Code) y LLOC (Logic Lines of Code, es decir, sin contar líneas de comentarios. Estos datos pueden obtenerse con Eclipse Metrics Plugin o mediante Sonarqube.
  - Clases de tests de las clases diseñadas (separadas por clases) y por cada clase de test correspondiente a una clase diseñada indicar SLOC y LLOC, y la suma de estos valores para todas las clases de test de una clase diseñada. Se indicará la ratio *code to test* calculada.
- Fecha límite de entrega: **20 de diciembre de 2019**. No se admitirán entregas fuera de plazo o por otra vía distinta a la indicada en estas normas.

Criterios de evaluación y recopilación de problemas frecuentes

En la Figura 1 se muestran los criterios que se aplicarán al evaluar esta práctica. A continuación se enumera una recopilación de problemas encontrados frecuentemente en este tipo de trabajos:

- No se consiguen altos niveles de cobertura de condición-decisión (complejidad).
- Demasiados tests para tan baja cobertura de clases poco complejas.
- El pipeline al hacer el publish en sonarqube no consigue que en sonarqube se muestre la cobertura alta como en EclEmma (si se ha usado esta herramienta), comprobar ambas cosas para ver por qué.
- Los tests en aislamiento no reproducen los tests que no son en aislamiento. Debe probarse lo mismo.
- No se consigue hacer funcionar los mocks.
- No se aplica "rama por tarea".
- Los objetivos ant no funcionan bien o varios objetivos ant hacen lo mismo.
- En el pom.xml (maven) no se definen profiles que permitan ejecutar diferentes tipo tests (ej: mvn test -P Isolation).
- No se se aprecia una buena realización de *pair programming*: no se observa cambio de rol a lo largo de la tarea de programación o la tarea se asigna sólo a un desarrollador y no a ambos en el issue tracker.

General			
PairProg	0,5		Haciendo Pair Programming a lo largo de la tarea. Se aprecia cambio de rol
IntegraciónContinua	5		Control de versiones, maven, ant, gitlab CI pipeline. Se desglosa en 5 items
QA	0,5		Se analiza la calidad usando SonarQube
Tests	4		Tests:Green, cobertura, mocks. Se desglosa en 3 items
	10		
Desglose de Integración Continua			
IC1	0,5		Se usa control de versiones en gitLab con respaldo en bitbucket. Es inmediato hacer funcionar el proyecto en otro entorno
IC2	1		Gestión de la Configuración: Control de Versiones: Ramificaciones. Issues: Se gestionan issues con estimación de tiempo e indicación de tiempo empleado
IC3	2		Gestión del proyecto y de dependencias con maven
IC4	1		Objetivos ant, se consiguen los objetivos ant basados en maven de forma que se sigue consiguiendo la gestión de dependencias automatizada y la independencia del entorno
IC5	0,5		gitlab CI, pipeline
	5		
Desglose de Tests			
tests1	2		Green. Todos los tests en verde (Calidad en los tests)
tests2	1		Cobertura. Se consigue alto nivel de cobertura aportando nuevos tests de caja blanca. Se busca cobertura de condición-decisión (complejidad)
tests3	1		Mocks: Se aportan tests en aislamiento basados en mocks
	4		

			PairProg	IC1	IC2	IC3	IC4	IC5	QA	tests1	tests2	tests3	check
muy bien	A	10	0,5	0,5	1	2	1	0,5	0,5	2	1	1	10
bien	B	7,5	0,375	0,375	0,75	1,5	0,75	0,38	0,375	1,5	0,75	0,75	7,5
regular	C	5	0,25	0,25	0,5	1	0,5	0,25	0,25	1	0,5	0,5	5
mal	D	2,5	0,125	0,125	0,25	0,5	0,25	0,13	0,125	0,5	0,25	0,25	2,5
muy mal o inexistente	E	0	0	0	0	0	0	0	0	0	0	0	0

Figura 1: Criterios de evaluación de la Práctica 3