PRINCIPIOS DE DESARROLLO DE SOFTWARE

2019-2020

Universidad Carlos III de Madrid

Ejercicio Guiado - 3

2019/2020

DESARROLLO DIRIGIDO POR PRUEBAS

Ejercicio Guiado 3

Universidad Carlos III de Madrid. Escuela Politécnica Superior

Autor: Javier García Guzmán

Sección 1

Objetivos

Este ejercicio guiado tiene como objetivos:

1) Poner en práctica el proceso de desarrollo dirigido por pruebas.
2) Aprender las herramientas necesarias para el desarrollo dirigido por pruebas.
3) Poner en práctica las técnicas de prueba funcionales y estructurales.

Caso Práctico

El propósito específico de este ejercicio consiste en definir, automatizar y ejecutar las pruebas unitarias necesarias que se utilizarán para verificar que el componente de gestión de tokens que Transport4Future se encuentra desarrollando para gestionar la comunicación segura entre los sensores y actuadores de un vehículo autónomo con los componentes responsables de su control.

Se proporciona una versión del código fuente que se está comenzando a desarrollar (se trata del código proporcionado en el EG2), aunque es necesario avisar que este componente, en este momento, no implementa funcionalidad alguna, solamente se proporciona para facilitar la aplicación de la técnica de desarrollo dirigido por pruebas (TDD) mediante la herramienta de automatización JUnit.

Es responsabilidad de los estudiantes realizar el desarrollo de las funcionalidades propuestas partiendo del código fuente proporcionado aplicando las técnicas de TDD y de diseño de casos de prueba según correspondan.

Los requisitos funcionales establecidos para las funciones del componente para las que se desea definir los casos de prueba funcionales y estructurales son los siguientes:

TM-RF-01: El componente proporcionará una función para permita generar una solicitud de token a partir de los datos que identifican al dispositivo que desea conectarse a la red del vehículo autónomo

- TM-RF-01-E1: Ruta del fichero con los datos necesarios de la máquina para la cual se quiere generar una nueva licencia. El fichero de entrada debe cumplir el siguiente formato:

```
{
    "Device Name":"<Person Name (20)>",
    "Type of Device":"<Sensor|Actuator>",
    "Driver Version":"<digits sequence including '.' in between not larger than 25 items>",
    "Support e-mail":"<Valid e-mail address>",
    "Serial Number":"<Combination of letters, numbers and bars without spaces>",
    "MAC Address":"<Valid MAC Address >"
}
```

- TM-RF-01-O1: El componente verificará la corrección de los datos proporcionados en el fichero.
- TM-RF-01-O2: En caso de que los datos sean válidos, codificará los datos recibidos utilizando un algoritmo MD5 y una contraseña que se defina. Esta codificación generará una cadena de caracteres que constituirá el valor del Token Request.
- TM-RF-01-S1: Un string correspondiente con el valor del Token Request generado.
- TM-RF-01-S2: Un mensaje de error ante las siguientes situaciones:
    - No se encuentra el fichero con los datos de entrada
    - El fichero de entrada no contiene los datos o el formato esperado
    - Se ha producido un error interno en la generación del Token Request.

La definición del método de la interfaz del componente que proporcionará esta funcionalidad es la siguiente:

```
String TokenRequestGeneration (String InputFile) throws TokenManagementException;
// String represents TM-RF-01-S1
// String InputFile represents TM-RF-01-E1
// TokenManagerException represents TM-RF-01-S2
```

TM-RF-02: El componente proporcionará una función para solicitar un token que permita identificar a un dispositivo como autorizado para interactuar con otros elementos del vehículo.

- TM-RF-02-E1: Ruta del fichero con los datos necesarios de la máquina para la cual se quiere generar una nueva licencia. El fichero de entrada debe cumplir el siguiente formato:

```
{
   "Token Request":"<String having XX characters>",
   "Notification e-mail":"<Valid e-mail address>",
   "Request Date":"<Valid date according to the following format dd/mm/yyyy HH:MM:SS>"
}
```

- TM-RF-02-O1: El componente deberá preparar la estructura lógica del token que consistirá en los siguientes campos:
    - **Header**, que a su vez está compuesto por dos campos, el primero "alg" que es un string que indica el tipo de algoritmo que se ha utilizado para firmar el token cuyo valor, por ahora, será "HS256" aunque en futuras versiones se admita otro valor. El segundo campo será "typ" que informa del tipo de token generado que, por ahora, solo admitirá un valor "PDS".
    - **Payload**, que a su vez estará compuesto por un campo que se corresponde con la cadena de caracteres de token request (device) junto con la fecha de emisión del token (issued at) y la fecha de expiración de este (expiration date).
    - **Signature** que es la firma utilizando el algoritmo antes mencionado "HS256" de los elementos incluidos tanto en el campo Header como en el Payload.
- TM-RF-02-O2: El componente deberá almacenar el token generado para poder verificar las peticiones que se realicen posteriormente.
- TM-RF-02-S1: Un string correspondiente a la codificación en base64urlencoding de los tres elementos que se corresponden con el token generado.
- TM-RF-02-S2: Un mensaje de error ante las siguientes situaciones:
    - No se encuentra el fichero con los datos de entrada.
    - El fichero de entrada no contiene los datos o el formato esperado.
    - Se ha producido un error interno en la generación del token.
    - Se ha producido un error interno en el almacenamiento del token.

- Se ha producido un error interno en la codificación del token.

La definición del método de la interfaz del componente que proporcionará esta funcionalidad es la siguiente:

```
String RequestToken (String InputFile) throws TokenManagementException;
// String represents TM-RF-01-S1
// String InputFile represents TM-RF-01-E1
// TokenManagerException represents TM-RF-01-S2
```

TM-RF-03: El componente de gestión de tokens permitirá verificar la validez de un token recibido como entrada.

- TM-RF-03-E1: Cadena de caracteres codificada en base64urlencoding correspondiente con el token que se desea validar.
- TM-RF-03-O1: El componente deberá descodificar todos los elementos del token (header, payload y signature) a partir del string recibido.
- TM-RF-03-O2: El componente deberá verificar que la fecha de expiración del token no se ha sobrepasado.
- TM-RF-03-O2: El componente deberá verificar que el token se corresponde con uno que tengamos registrado para conceder acceso al servicio.
- TM-RF-03-S1: Un valor booleano que indique si el token es válido o no.
- TM-RF-03-S2: Un mensaje de error ante las siguientes situaciones:
    - La cadena de caracteres de la entrada no se corresponde con un token que se pueda procesar.
    - No se encuentra registrado el token para el cual se solicita verificación.

La definición del método de la interfaz del componente que proporcionará esta funcionalidad es la siguiente:

```
boolean VerifyToken (String Token) throws TokenManagementException;
// boolean represents TM-RF-03-S1
// String Token represents TM-RF-03-E1
// TokenManagerException represents TM-RF-03-S2
```

Sección 2

Análisis de clases de equivalencia y valores límite

Los pasos para aplicar las técnicas de análisis de clases de equivalencia y valores límite se presentan a continuación. Esta técnica se aplicará para la prueba de la funcionalidad 1.

Paso 2.1: Identificar las clases de equivalencia

La identificación de las clases de equivalencia se hace a partir de las entradas/salidas recibidas y proporcionadas por el método objeto de prueba.

Una clase de equivalencia consiste en un conjunto de datos que se tratan de igual manera por un método o deberían proporcionar el mismo resultado.

Dependiendo de la naturaleza de los parámetros entrada y salidas, hay varias reglas que se deben tener en cuenta. Estas reglas están relacionadas con:

- Rango de valores de entrada / salida.
- Número de valores de entrada.
- Semántica de los valores de entrada.

Paso 2.2: Identificar los valores límite a considerar

Dependiendo de la naturaleza de los parámetros de entrada, aplicar las reglas que se consideren oportunas para identificar los valores límite a considerar.

Paso 2.3: Identificar los casos de prueba a considerar

De acuerdo con la información de las clases de equivalencia y valores límite en los dos pasos anteriores, el estudiante debe definir los casos de prueba que será necesario considerar. Las heurísticas para identificar los casos de prueba necesario son:

- Se pueden combinar clases de equivalencia válida en un único caso de prueba si es posible.
- Cada clase de equivalencia inválida y valor límite relevante deberá considerarse en un caso de prueba independiente.

La información a incluir en la especificación de cada caso de prueba es la siguiente:

- Identificador
- Valores de entrada
- Resultados esperados

Paso 2.4: Implementar los casos de prueba definidos

El estudiante tiene que programar los casos de prueba identificados en el paso anterior utilizando el framework y las funciones incluidas en la API de JUnit 5.

Es obligatorio comentar el código correspondiente con los casos de prueba generados utilizando la siguiente plantilla:

```
/* Caso de Prueba: <Id - Nombre>
* Clase de Equivalencia o Valor Límite Asociado: <Valor>
* Técnica de prueba: <Clases de Equivalencia | Valor Límite>
* Resultado Esperado: <Descripción>
*/
```

En caso de que ya existan versiones del código funcional y del código de prueba registrado en GIT al finalizar sesiones de trabajo anteriores, será necesario asegurar que se dispone de la última versión del código registrada en el repositorio.

Paso 2.5: Ejecutar los casos de prueba desarrollados

El estudiante tiene ejecutar el caso de prueba generado utilizando la interfaz que proporciona el entorno de JUnit.

Como, en principio, no se habrá generado código funcional alguno para superar esta prueba, el resultado de la prueba debe ser Fallo (Failure).

Paso 2.6: Implementar el código funcional asociado a la prueba

El estudiante deberá implementar el código que permita la correcta ejecución del caso de prueba.

Una vez finalizado este paso, será necesario repetir los pasos 2.3-2.5 con el resto de los casos de prueba que se vayan identificando mediante la técnica de análisis de clases de equivalencia y valores límite.

Este proceso se repetirá hasta identificar e implementar todos los casos de prueba y conseguir la correcta implementación de la funcionalidad 1 del componente reutilizable.

Paso 7: Registro del código funcional y de pruebas en Git

El estudiante deberá crear una Branch llamada EG3 al inicio del ejercicio y deberá ir dejando en esta rama el ejercicio guiado 3.

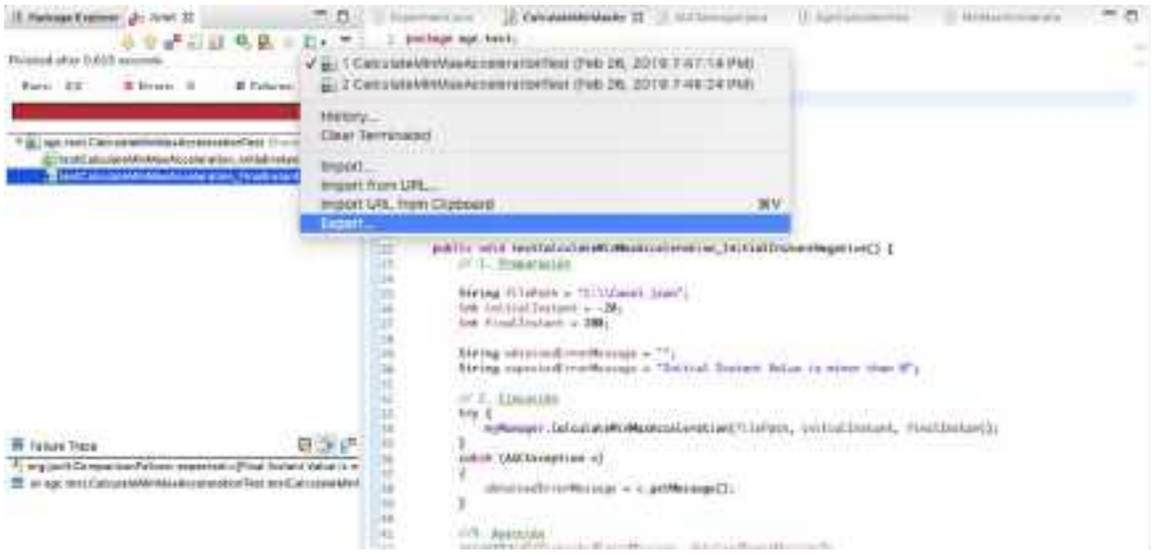
Como la resolución del ejercicio guiado supondrá diversas sesiones de trabajo, cada sesión debe finalizar con la correcta implementación de un caso de prueba (el que corresponda en cada momento).

Al finalizar la sesión de trabajo se deberá realizar en GIT la ejecución los comandos *Commit* y *Push* para el correcto registro del trabajo realizado, de acuerdo con las instrucciones proporcionadas en el Ejercicio Guiado 2.

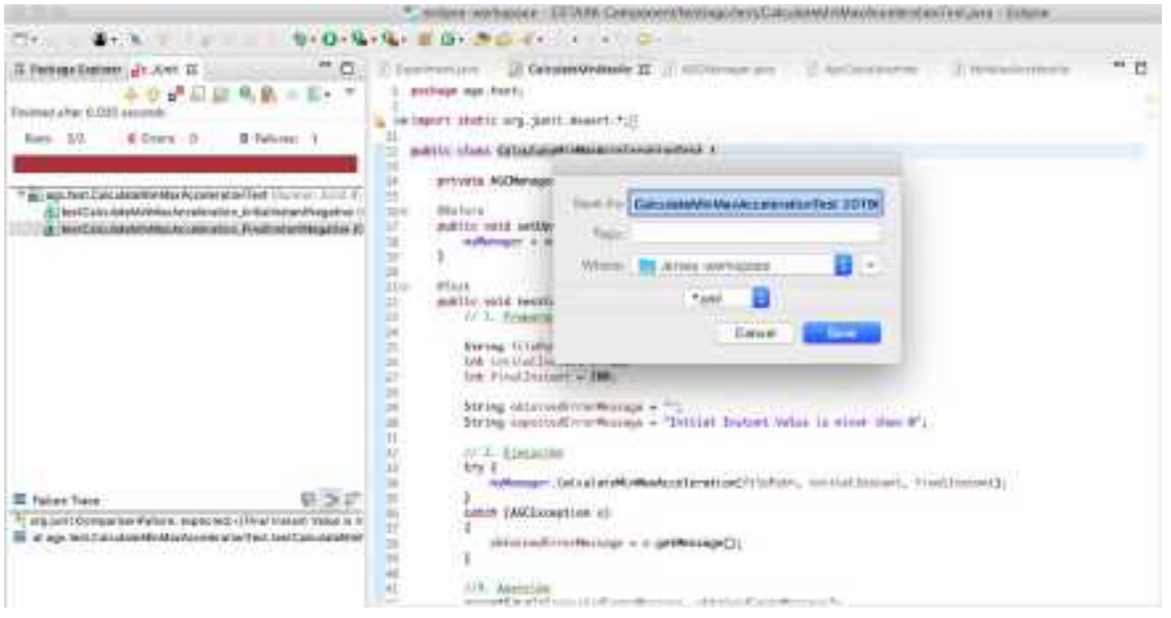
Es necesario recordar que también hay que registrar en GIT el registro en XML que evidencia la correcta ejecución de los casos de prueba definidos en el momento de registrar el código en el sistema de control de versiones.

Este registro se obtiene de la siguiente manera:

1) Pulsando el Botón Test Run History en la interfaz de ejecución de JUnit y seleccionando la opción Export.



2) Asignando como nombre del registro: <*Nombre de la clase que incluye los casos de prueba*>-*Registry.xml*.



Sección 3

Análisis Sintáctico

Los pasos para aplicar la técnica de análisis sintáctico se presentan a continuación. Estas técnicas se aplicarán para la prueba de la funcionalidad 2.

Paso 3.1: Definición de la gramática

El estudiante tiene que definir una gramática para representar los valores esperados en el fichero de entrada al que se hace referencia en el parámetro de entrada de cada una de las funciones que se corresponden con los requisitos funcionales a probar.

Para ello, es necesario considerar que:

- La gramática debe ser de tipo 2 ó 3, es decir, regular e independiente del contexto.
- Es necesario que todas las producciones tengan un único símbolo terminal en la parte izquierda de la misma.
- No se deben utilizar símbolos vacíos (Lambda) en las producciones de la gramática.

Paso 3.2: Creación del árbol de derivación

Se debe crear un árbol de derivación a partir de la gramática obtenida durante el paso anterior. Cada símbolo (terminal o no) se debe incluir en un nodo diferente. Los nodos se numerarán comenzando por el 1, de arriba a abajo y de izquierda a derecha.

Se debe diferenciar fácilmente los niveles del árbol y, en cada uno de ellos, si los nodos que se incluyen son terminales o no.

Paso 3.3: Identificar los casos de prueba a considerar

En el alcance de este ejercicio, el estudiante debe considerar tanto los posibles casos de prueba válidos como los inválidos:

Par los casos válidos, todos los nodos no terminales deben estar representados en, al menos, un caso de prueba.

Para identificar los casos de prueba inválidos. es necesario reducir los posibles casos a generar considerando los siguientes pasos:

Paso 1 – Para los nodos no terminales, proceder a su omisión o adición.

Paso 2 – Para los nodos terminales, proceder a su modificación, introduciendo cambios en los valores esperados.

Paso 3.4: Implementar el código de prueba y funcional

El estudiante tiene que programar el código de prueba y el código funcional siguiendo el mismo proceso que el establecido en los pasos 2.3 al 2.6 pero considerando los casos de prueba que se deriven exclusivamente del análisis sintáctico de las entradas de la funcionalidad 2.

Es obligatorio comentar el código correspondiente con los casos de prueba generados utilizando la siguiente plantilla:

```
/* Caso de Prueba: <Id - Nombre>
* Nodo/s del Árbol de Derivación: <Valores específicos>
* Tipo de Prueba: <Valor Normal | Omisión | Repetición | Omisión>
* Técnica de prueba: Análisis Sintáctico
* Resultado Esperado: <Descripción>
*/
```

Sección 4

Técnicas de prueba estructurales

Los pasos para aplicar la técnica de complejidad ciclomática de McCabe, análisis sintáctico se presentan a continuación. Estas técnicas se aplicarán para la prueba de la funcionalidad 3.

Paso 4.1: Definición del grafo del flujo de control del método a probar

El estudiante tiene que definir un grafo que represente el flujo de control interno del método *VerifyToken*.

Paso 4.2: Identificar los casos de prueba utilizando la técnica de los caminos básicos

Para cada uno de los métodos anteriormente considerados se deben identificar casos de prueba considerando que:

- En primer lugar, es necesario calcular el valor de la complejidad ciclomática de McCabe para saber cuántos caminos básicos se deben contemplar en los casos de prueba a considerar.
- Posteriormente se debe identificar cada uno de esos caminos básicos, cada uno de ellos, se creará un único caso de prueba.
- A continuación, se deben establecer las entradas necesarias para que el método ejecute cada camino básico identificado.
- Por último, se deben identificar las salidas esperadas de cada uno de los casos de prueba identificados.

Paso 4.3: Identificar los casos de prueba utilizando la recomendación para la prueba de bucles

Este paso consiste en la identificación de los casos de prueba necesarios para probar que los bucles incluidos en el código fuente de los métodos anteriormente considerados están implementados correctamente. Para ello es necesario utilizar las reglas y recomendaciones discutidas durante las clases de teoría relativas a las técnicas de prueba estructurales.

Para cada caso de prueba es necesario identificar:

- El propósito del caso de prueba, indicando las comprobaciones que se realizarán de cada uno de los bucles incluidos en el método a probar.
- Las entradas necesarias para que el método ejecute cada camino básico identificado.
- Las salidas esperadas de cada uno de los casos de prueba identificados.

Paso 4.4: Implementar el código de prueba y funcional

El estudiante tiene que programar el código de prueba y el código funcional siguiendo el mismo proceso que el establecido en los pasos 2.3 al 2.6 pero considerando los casos de prueba que se deriven exclusivamente de las técnicas de prueba estructurales consideradas en los pasos 4.2 y 4.3 para la funcionalidad 3.

Es obligatorio comentar el código correspondiente con los casos de prueba generados utilizando la siguiente plantilla:

```
/* Caso de Prueba: <Id - Nombre>
* Técnica de prueba: Análisis Estructural - <Caminos Básicos – Pruebas de Bucles>
* Resultado Esperado: <Descripción>
*/
```

Sección 5

Reglas y Procedimientos

Entrega

Este ejercicio se completará en parejas.

Una vez que se finalice el ejercicio, el estudiante deberá enviar un fichero que contenga:

1) Documento en formato PDF que contenga:
   a) El análisis de clases de equivalencia y valores límites para la funcionalidad 1.
   b) La gramática y el árbol de derivación para la funcionalidad 2 que se debe probar utilizando la técnica de análisis sintáctico.

   Este documento se registrará en el repositorio de Git del proyecto, creando un folder que se denominará "*Documentación*".

2) Se debe registrar en el repositorio de Git que el profesor de prácticas indique para este ejercicio guiado.
   a) El código funcional que los alumnos han generado durante el proceso de desarrollo dirigido por pruebas considerado en el ejercicio guiado.
   b) El código de prueba en JUNIT para la automatización de los casos de prueba considerados.
   c) Un informe generado en XML por JUnit que incluya los resultados de la ejecución de los casos incluidos para cada una de las clases de prueba que contienen los casos considerados para cada una de las funcionalidades y técnicas consideradas.

La fecha límite para la entrega del ejercicio es 02/04/2019 antes de las 23:59.

De acuerdo con las normas de evaluación continua en esta asignatura, si un estudiante no envía la solución del ejercicio antes de la fecha límite, el ejercicio será evaluado con 0 puntos.

Cada estudiante debe guardar una copia de la solución entregada hasta la publicación de las calificaciones finales de la asignatura.