Guía de desarrollo, Anexo 03.04

Normativa de Pruebas Unitarias

Autor: Oficina de Pruebas

GERENCIA INFORMÁTICA
JOSEFA VALCÁRCEL, 44
28027-MADRID

Índice General

1 Introducción

1.1 Objetivo

El objetivo de este documento es establecer las normas para el diseño, construcción y ejecución de pruebas unitarias en los proyectos de la Gerencia de Informática de la Dirección General de Tráfico.

1.2 Audiencia

Este documento está dirigido a todas las personas que colaboren en labores relacionadas con la gestión, desarrollo, auditoría, implantación y explotación de los sistemas de información de la gerencia de informática de la Dirección General de Tráfico.

2 Pruebas unitarias

Una prueba unitaria es un método por el cual se prueba una unidad de código o conjunto de líneas de código que satisfacen un requisito, teniendo como objetivo encontrar fallos.

Una prueba unitaria se caracteriza por:

- Validar pequeñas cantidades de código, es decir, solamente prueba el código de un requisito específico.
- No interfiere con el código desarrollado por el resto del equipo de trabajo.
- Es repetible y predecible. No importa el orden y las veces que se repita la prueba, el resultado siempre debe de ser el mismo.

2.1 Buenas prácticas

A continuación se detallan las características que debe tener una prueba unitaria para que sea correcta:

- Atómica: significa que el test prueba la mínima cantidad de funcionalidad posible.
- Independiente: significa que un test no puede depender de otros para producir un resultado satisfactorio y no deben lanzarse en orden. Cualquier comportamiento dado debe corresponderse con un único test. Esto es así para que, en caso de modificar una funcionalidad, solo un test se vea afectado.
- Inocua: significa que no altera el estado del sistema. Evitar construir test que produzcan efectos secundarios, ya que puede afectar a la ejecución de otros test y no son reproducibles de forma automática.
- Rápida: ya que al ejecutar un gran número de tests cada pocos minutos se ha demostrado que la espera puede resultar improductiva.

Si no cumple estas premisas entonces no se considera una prueba unitaria.

2.2 Normativa

A continuación se detalla la normativa de las pruebas unitarias en JUnit:

- El código fuente de los test unitarios de desarrollo del proyecto deberán localizarse en: **src/test/java.**
- El código fuente de los test unitarios de integración del proyecto deberán localizarse en: **src/it/java.**
- Los archivos de configuración necesarios se encontrarán en: **src/test/resources.**
- Se debe utilizar el mismo paquete que las clases implementadas, de modo que sea más sencillo identificar y localizar la funcionalidad probada. Por ejemplo, para las clases pertenecientes al paquete **es.trafico.apli.inter.alta** localizadas en **src/main/java**, las clases de test se construirán bajo el paquete **es.trafico.apli.inter.alta** en **src/main/test.**
- Para las clases de test, se propone utilizar la nomenclatura **[nombreClase]Test.java**, donde nombreClase será el nombre de la clase a probar. Por ejemplo, para la clase MensajeLog.java su clase de test será MensajeLogTest.java.
- Para los métodos de test, se propone utilizar un nombre que describa la funcionalidad probada. Por ejemplo, testLogMensajeVacio(), testLogMensajeNulo(),…
- No inicializar usando el constructor de la clase, utilizar la capacidad que ofrece JUnit para inicializar un test sobrescribiendo el método setUp(). Esta práctica permite obtener una traza más detallada si se produce una excepción en el momento de inicialización del test.
- No utilizar rutas absolutas sino rutas relativas, por ejemplo, para cargar datos desde ficheros.
- Evitar que los test sean dependientes del tiempo que utilicen datos susceptibles de expirar en una fecha determinada.
- Considerar que puede haber variables dependientes del sistema cuando se implementa el test, pudiendo hacer que el test falle en distintos entornos. Por ejemplo: locale,…
- No capturar excepciones en los test, ya que JUnit provee captura de excepciones haciendo que el test falle. La captura manual de excepciones puede provocar falsos positivos en la ejecución de los test.

- Documentar los test de forma apropiada utilizando JavaDoc y evitando comentarios dentro del código.

A continuación se muestra una plantilla de test en JUnit que permite la generación automática de los métodos de prueba:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<templates>
<template autoinsert="false" context="java-members" deleted="false" description="JUnit 4
test case set up method" enabled="true" name="setUp">@${testType:newType(org.junit.Before)}
public void setUp() { ${cursor} }</template>
<template autoinsert="false" context="java-members" deleted="false" description="JUnit 4
test case clean up method" enabled="true"
name="tearDown">@${testType:newType(org.junit.After)} public void tearDown() { ${cursor}
}</template>
<template autoinsert="false" context="java-members" deleted="false" description="JUnit 4
test method" enabled="true" id="org.eclipse.jdt.ui.templates.test_junit4"
name="test">@${testType:newType(org.junit.Test)} public void test${Name}() { ${cursor}
}</template>
<template autoinsert="false" context="java-members" deleted="false" description="Data
collection method for parameterized JUnit test cases based on an enum type" enabled="true"
name="parametersOverEnum">${a:importStatic(com.google.common.collect.Lists.newLinkedList)}$
{b:import(java.util.Collection)}@${annotation:newType(org.junit.runners.Parameterized.Param
eters)} public static Collection<Object[]> data() { Collection<Object[]> result =
newLinkedList(); for( ${EnumType} ${element} : ${EnumType}.values() ) { result.add( new
Object[] { ${element}${cursor} } ); } return result; }</template>
<template autoinsert="false" context="java-members" deleted="false" description="JUnit 4
rule definition" enabled="true" name="rule">@${annotationType:newType(org.junit.Rule)}
public ${final} ${ruleType} ${variableName} = new ${ruleType}();${cursor}</template>
<template autoinsert="false" context="java-members" deleted="false" description="JUnit 4
test method with expected setting" enabled="true"
name="testExpected">@${testType:newType(org.junit.Test)}( expected = ${throwableType}.class
) public void test${Name}() { ${cursor} }</template>
      </templates>
```

2.3 Indicadores de calidad de pruebas unitarias

El grado de cumplimiento mínimo de las pruebas unitarias será el siguiente:

- Para proyectos nuevos se exigirá un mínimo de un 80 % de cobertura de código.
- Para proyectos evolutivos se exigirá un mínimo de un 80 % de cobertura de código en el código nuevo y modificado.

Esto aplica a todos los proyectos arrancados con la Guía de desarrollo 2018 (V03_v01_R001). Si los equipos de desarrollo no han incluido pruebas unitarias desde el inicio del primer sprint, las

pruebas de los sprints que no hayan sido incluidas, se irán añadiendo de forma incremental de manera retroactiva a lo largo de los sprints posteriores, añadiendo a las pruebas de cada sprint en curso, un % de pruebas de los sprints previos, hasta alcanzar el 80% de cobertura de código.

Ejemplo de cobertura de código de un proyecto de 5 sprints que empieza a incluir sus pruebas en el sprint 2

Sprint	1	2	3	4	5
Cobertura de código		50%	60%	70%	80%
		(1)	(2)	(3)	(4)

(1) 50% de cobertura incluyendo los casos de los 2 primeros sprints
(2) 60% de cobertura incluyendo los casos de 3 sprints
(3) 70% de cobertura incluyendo los casos 4 sprints
(4) 80% de cobertura incluyendo los casos de todo los sprints

2.4 Incorporación de las pruebas unitarias en el ciclo de integración continúa

Teniendo en cuenta la importancia que tiene la ejecución de las pruebas unitarias con frecuencia, estas deben automatizarse y ejecutarse cada vez que se realice la compilación de un proyecto. Para ello es necesario la integración de JUnit con la herramienta de construcción de proyectos de DGT (Ver anexo adecuado).