INFORME DE PRUEBAS DE SISTEMAS DE INFORMACIÓN WEB

28 DE FEBRERO DE 2022

E1.07

https://github.com/jaimemosber/Acme-Toolkits

Jaime Borrego Conde: jaiborcon@alum.us.es

Antonio Campos Gil: antcamgil@alum.us.es

Ana Conde Marrón: anaconmar@alum.us.es

Gonzalo Martínez Fernández: gonmarfer2@alum.us.es

Jaime Moscoso Bernal: jaimosber@alum.us.es

Enrique Muñoz Pérez: enrmunper@alum.us.es

Tabla de contenido

Resumen ejecutivo

El objetivo del presente documento es exponer a los profesores de la asignatura los conocimientos que el grupo de trabajo ha adquirido a través de formación previa y que actualmente se posee acerca del *testing* de los sistemas de información web, pues sobre este será sobre los que se trabaje en la asignatura Diseño y Pruebas II.

Tabla de revisión

Número de revisión	***Fecha***	***Descripción***
1	*19/02/2022*	*Firma de acuerdos*

Contenido

Introducción

Cualquier sistema web debe ser testeado: es la forma más eficiente de detectar *cruft* y no aumentar la deuda técnica. Además, permite cerciorarse de que el código no contiene fallos de una forma más rápida que sin pruebas.

La realización de test en sistemas de información web puede ser una tarea compleja y muy tediosa si no se automatiza. Sin embargo, para comprender el *testing*, primero hay que conocer dos conceptos fundamentales:

- Verificación. ¿El producto cumple las especificaciones demandadas?
- Validación. ¿Es lo que el cliente quiere?

Por tanto, de manera simplificada, el *testing* se define como la ejecución de código con diversos datos de entrada para comprobar que el *software* se comporta como se espera que lo haga, es decir, que cumpla las especificaciones propuestas y que sea exactamente lo que el cliente quiere.

En cuanto al *testing*, la unidad de medida mínima es el sujeto bajo pruebas (SUT, Subject Under Testing), que depende del tipo de lenguaje:

- Si se utiliza lenguaje funcional, la unidad mínima son las funciones.
- Si se utiliza un lenguaje orientado a objetos, el SUT es la interfaz pública de cada clase.

El conjunto de test implementados para una aplicación se denomina suite de pruebas.

Cabe recalcar que nunca se es capaz de probar una aplicación al 100%, puesto que no existen ni los recursos ni el tiempo necesarios para ello, ya que hasta el *software* más simple requiere de un tiempo desmesurado para probar absolutamente todos los casos. Por lo tanto, los test no pueden ser exhaustivos, deben mostrar comportamientos irregulares o fallos (bugs), pero nunca la ausencia de ellos.

En general, se pueden encontrar 5 tipos distintos de pruebas:

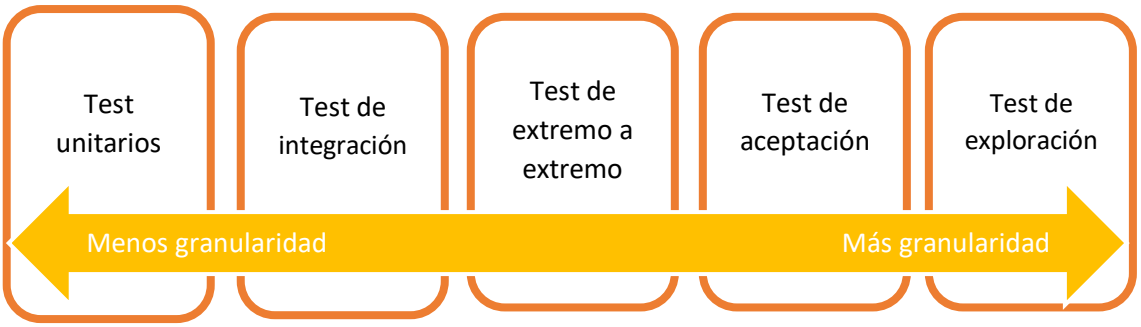

Ilustración 1. Tipos de pruebas

Esta taxonomía tiene distinta granularidad (cuán específico o general es), siendo las pruebas unitarias aquellas con menor granularidad y las pruebas exploratorias las que se encuentran en el extremo opuesto del espectro.

Pruebas Unitarias

Las pruebas unitarias se ejecutan muy rápido al tener menor granularidad y alcance, por tanto, la mayoría de las pruebas de una suite deberían ser unitarias. Usualmente, se suele realizar una clase de pruebas por cada clase realizada (en lenguaje orientado a objetos).

Las principales características de las pruebas unitarias se pueden resumir en el acrónimo FIRST, y son:

- (**F**) Rápidas: fáciles y rápidos de ejecutar.
- (**I**) Independientes: ninguna prueba debe de depender de precondiciones de otros test.
- (**R**) Repetibles: los test no pueden depender de factores externos como la fecha actual o constantes mágicas.
- (**S**) Autoevaluados: las pruebas tienen que comprobar estos mismos si han pasado o no con éxito en vez de depender de factores humanos.
- (**T**) En tiempo: deben ser creados y actualizados conforme se desarrolla el código.

Estructura

Todos los test deben seguir la siguiente estructura en tres pasos:

1. *Arrange*: configurar los datos de las pruebas.
2. *Act*: ejecutar el SUT.
3. *Assert*: comprobar que se cumplen los resultados con lo esperado.

Los datos utilizados para ejecutar los test se les denominan *fixture*.

Tipos

En general, encontramos dos tipos de prueba según los datos utilizados:

- Casos positivos: comprueban el "camino feliz", es decir, el comportamiento normal y esperado de la aplicación con datos que podrían ser reales.
- Casos negativos: comprueban cómo se comporta el sistema ante datos irregulares o situaciones imposibles en un contexto usual.

Podemos encontrar dos tipos de prueba según cómo se relaciona con su entorno:

- Test sociable: se prueba el SUT, pero utiliza el resto del sistema en caso de requerir información externa.
- Test aislado: se prueba el SUT, pero en caso de requerir datos externos se usan "dobles" que contienen los datos que se obtendrían de las peticiones.

Test Aislados

Dobles

Se hacen pasar por objetos reales, pero en realidad son versiones más rápidas o simples de los mismos. Se usan en los test aislados para evitar efectos secundarios no deseados si otra parte del sistema no funcionara correctamente y para evitar una preparación compleja.

Los dobles tienen los siguientes tipos principales:

- *Stubs*: no contienen lógica alguna, simplemente devuelven respuestas. Se usan para especificar un valor o colocar al sistema en un estado concreto.
- *Mocks*: se usan cuando no cambia el estado del sistema. Reemplazan la interacción entre objetos. Esperan a ser llamados de una forma concreta.
- *Fakes*: versiones ligeras de objetos reales que se comportan como ellos.
- *Dummies*: objetos que se pasan como parámetro, pero nunca se usan.
- *Spies*: *stubs* que guardan información sobre cómo son invocados.

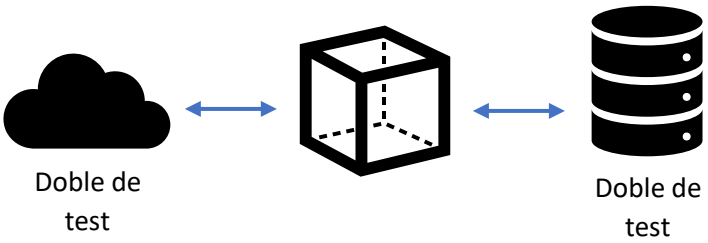

Ilustración 2. Dobles de prueba

Existen distintos *framework* para la implementación de dobles de pruebas, aunque los más conocidos son *Mockito* y *Web MVC.*

Costuras (Seam)

Se trata de un lugar donde se altera el comportamiento de un programa sin modificarlo realmente. En los test, se implementan a través de los dobles.

Test extremo a extremo

Consumen mucho tiempo y recursos, pues son los test más exhaustivos.

Automatización de pruebas

Por último, es importante recalcar la importancia de la automatización de las pruebas. Y es que el código evoluciona y avanza, y lo que funciona hoy puede no hacerlo mañana. Si no se automatizan las pruebas, es probable que haya que estar constantemente arreglando tanto test como código, lo que, en última instancia, provocaría que las personas encargadas de ello cometiesen fallos, bien por pereza, bien por aburrimiento. Uno de los *framework* más populares para la realización de pruebas en Java es JUnit5.

Imagen 1: Framework Junit5

Algunas buenas prácticas a la hora de realizar test son:

- Llevar a cabo test unitarios parametrizados.
- Usar los métodos de *Fluent Assertions*.
- Mantener los test unitarios centrados en una sola tarea.
- Separar la causa y el efecto de los test.
- Seguir el principio DAMP (frases descriptivas y con valor), en lugar del principio DRY (no te repitas), más adecuado para el código que para las pruebas.

¿Cuántos test son necesarios?

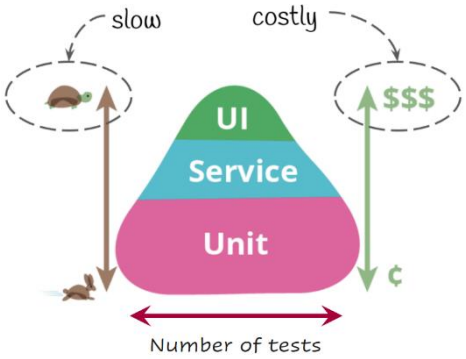

Imagen 2: Pirámide de tests

Para responder a esta pregunta, se deben tener en cuenta diversos factores. Si fallan test de niveles superiores, pero no de nivel inferior, posiblemente sea necesario incluir un nuevo test de nivel inferior. Además, los test deberán pertenecer siempre al nivel más bajo posible, mientras que los test de nivel superior solo se centrarán en aquellas tareas que no puedan realizar el resto. Asimismo, hay que asegurarse de que los test cubren todo el código del sistema, es decir, la cobertura debe ser máxima.

Estrategias de recubrimiento

Secuencias: cada secuencia es probada al menos una vez.

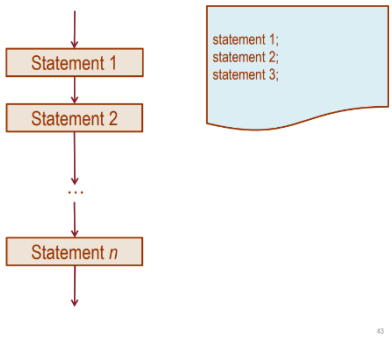

Imagen 3: Secuencia

Condicionales: cada rama de la sentencia condicional es ejecutada.

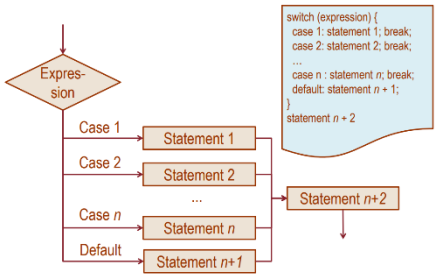

Imagen 4: Condicionales

Bucles: asegurarse de que el bucle no se pruebe, además de comprobar que se recorre

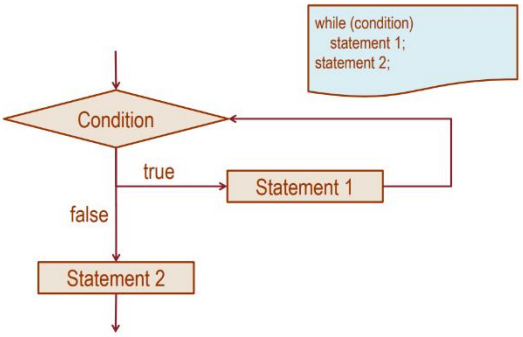

Imagen 5: Bucles

tanto una como varias veces.

Rangos: asegurarse de que cada atributo obtiene valores cerca de los límites del rango.

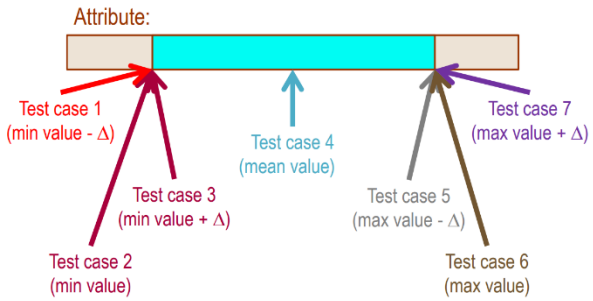

Imagen 6: Rangos

Opcionales: probar tanto con valores nulos como con valores no nulos.

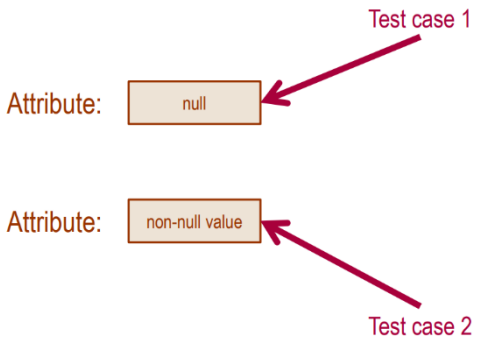

Imagen 7: Opcionales

Colecciones: asegurarse de que tienen cero, uno y más elementos.

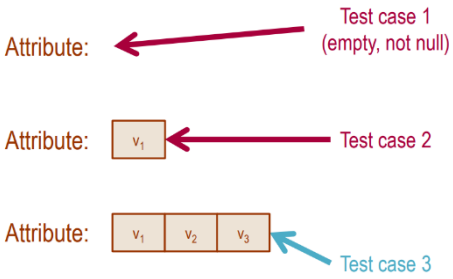

Imagen 8: Colecciones

Bibliografía

Intencionalmente en blanco.