**Testing de un WIS**

## **Grupo C1.048**



## **Integrantes del grupo**

* Basallote Braza, David: [davbasbra@alum.us.es](mailto:davbasbra@alum.us.es)
* Chaves Cumbreras, David: [davchacum@alum.us.es](mailto:davchacum@alum.us.es)
* González Vázquez, Guillermo: [guigonvaz@alum.us.es](mailto:guigonvaz@alum.us.es)
* Herrera Luján, Marco Antonio (manager): [marherluj@alum.us.es](mailto:marherluj@alum.us.es)
* Rodríguez Muñoz, Rafael: [rafrodmunn@alum.us.es](mailto:rafrodmunn@alum.us.es)

## **Repositorio**

<https://github.com/DP2-IIS-C1048/Acme-ANS-D>

06/03/2025

[Grupo C1.048 1](#_Toc1150009319)

[Integrantes del grupo 1](#_Toc711919)

[Repositorio 1](#_Toc238382923)

[Tabla de versiones 2](#_Toc967241493)

[Introducción 2](#_Toc1779180149)

[Importancia del testing y el diseño 3](#_Toc732397998)

[Vocabulario utilizado en el testing 3](#_Toc35147220)

[Tipos de testing 3](#_Toc370568362)

[Tests unitarios 4](#_Toc1881576437)

[Tests End to End y de Aceptación 4](#_Toc90141192)

[Tests sociables 4](#_Toc2077083026)

[Tests aislados 4](#_Toc417329256)

[Testing de un framework 4](#_Toc388932138)

[Test doubles 5](#_Toc1213566504)

[Cobertura de un test 5](#_Toc1345024038)

[Buenas prácticas en el testing 5](#_Toc1302150239)

[Conclusión 6](#_Toc1316341898)

[Bibliografía 6](#_Toc1481654012)

Resumen ejecutivo

En este documento se recoge toda la información relevante sobre conocimientos del testing de un WIS, definiendo brevemente en que consiste y su importancia. También se detallan los tipos de test conocidos y herramientas que ayudan a la hora de hacer test como los “test doubles”. Además, se detallan métodos para realizar pruebas de forma que no se pasen por alto casos de prueba. Por último, se dan un conjunto de buenas prácticas a la hora de llevar a cabo el testing.

Tabla de versiones

|  |  |  |
| --- | --- | --- |
| **Versión** | **Cambio** | **Fecha** |
| 1.0.0 | Inicio del documento | 06/03/2025 |

# Introducción

En este documento se exponen los puntos más importantes acerca del testing de un Sistema de Información Web aprendidos en asignaturas anteriores a Diseño y Pruebas II. Prácticamente, toda la información acerca del testing de un WIS se obtuvo en la asignatura Diseño y Pruebas I.

En el documento se encuentran varios apartados que explican el conocimiento obtenido en la asignatura mencionada:

Primero se define qué es el testing, centrándose en su aplicación directa al desarrollo de un Sistema de Información. Se destaca la importancia de realizar un buen proceso de comprobación del código para poder asegurar que el producto obtenido del desarrollo cumple efectivamente con los requisitos tanto funcionales como no funcionales del cliente y de dedicar un tiempo en el diseño a realizar un buen plan de pruebas que permita incluir esta tarea de forma orgánica dentro del propio desarrollo.

Posteriormente se proporciona un pequeño glosario con terminología útil a la hora de leer el documento y se definen los tipos de testing que se han aprendido mayoritariamente: testing unitario, la granularidad más fina en cuanto a realizar testing; testing end to end y de aceptación, tanto para comprobar la funcionalidad del sistema en su totalidad como para comprobar que se cumplen los requisitos del cliente; testing sociable, el testing que se realiza utilizando colaboradores reales junto con el SUT y testing aislado, en el que se utilizan colaboradores ficticios o doubles para comprobar el SUT.

También se define cómo se comprueba el código cuando se utiliza un framework y los test doubles más utilizados en otras asignaturas.

Por último, se definen las claves más importantes aprendidas para realizar testing en cuanto a la cobertura de las pruebas, es decir, el porcentaje del código no trivial realizado que está comprobado que funciona y a las buenas prácticas cuando se realiza testing para que los tests sean organizados, fáciles de mantener y explícitos en cuanto a su función.

El documento se estructura para facilitar una introducción a conceptos de testing más generales, seguido de los tipos de testing aprendidos y, por último, define conceptos que marcan la diferencia a la hora de comprobar el funcionamiento de un software.

Definición de testing

El testing de un WIS consiste en comprobar el correcto funcionamiento de nuestro sistema. Entiéndase como correcto funcionamiento que cumple con los requisitos de nuestro cliente y que no genera errores inesperados.

# Importancia del testing y el diseño

Es importante verificar que el producto software que hemos construido cumple con los estándares de calidad requeridos por el cliente, además de funcionar como se espera que funcione.

La única manera de asegurar que un producto software cumple con los requisitos anteriormente mencionados es mediante la realización de pruebas de testing del mismo.

Como el testing es una parte tan importante del desarrollo software, es conveniente incluir un plan de pruebas específico para nuestro proyecto durante la etapa de diseño.

# Vocabulario utilizado en el testing

* SUT: Son las siglas de Subjet Under Test, es decir, el sujeto que vamos a probar.
* Bug: Es una característica de nuestro sistema que provoca errores. Mientras más se tarde en encontrar el bug, mayor podrá ser su impacto en el sistema.
* Código trivial: Es el código que no merece la pena ser comprobado, ya sea por su falta de complejidad o por ser perteneciente al propio lenguaje de programación (por ejemplo, comprobar que funciona el método get de un atributo)
* Test double: es un objeto que sirve para reemplazar a un objeto con características reales por un objeto ficticio (con características inventadas) en un test.

# Tipos de testing

Existen varias formas de probar un sistema. Unas se enfocan en probar trozos pequeños del código y otras abarcan un mayor rango. Conocemos 5 tipos de tests en función del código abarcado en cada uno:

* Test unitarios
* Test de integración
* Tests End to End
* Test de aceptación
* Test de exploración

También existen distintos tipos de test en función de si los mismos se comunican con clases con las que se comunicarían en un entorno de producción o no:

* Tests sociables
* Tests aislados

A continuación, se dan más detalles sobre los tipos de test con los que se ha trabajado previamente.

## Tests unitarios

Los test unitarios proveen el enfoque más pequeño de todos los tests. Al sujeto de prueba se le llama unidad. Estos test deben ejecutarse de forma rápida y, normalmente, los colaboradores externos del SUT son remplazados con dobles. La mayoría de los test de un sistema deberían ser test unitarios.

## Tests End to End y de Aceptación

Los tests End to End verifican que todos los componentes del sistema funcionan correctamente en conjunto, mientras que los tests de aceptación comprueban que se cumplen los requisitos del cliente, es decir, que el sistema hace lo que se le pide.

Hay dos tipos principales de tests End to End

* Tests End to End externos: Se despliega la aplicación entera en un servidor de prueba y ejecutamos los tests de forma externa a la aplicación, enviando peticiones HTTP y verificando las respuestas obtenidas.
* Tests End to End internos: Se despliega igualmente la aplicación entera en un servidor de prueba, pero se simulan las llamadas HTTP que se envían al sistema. Esta técnica es normalmente utilizada para políticas de control de acceso o seguridad del sistema.

Normalmente este tipo de tests consumen mucho tiempo y recursos, por lo que es importante no abusar de ellos y ejecutarlos con mucha menor frecuencia que otro tipo de tests (por ejemplo, los tests unitarios).

## Tests sociables

Los tests sociables son aquellos que permiten colaboradores “reales”, es decir, que permiten la interacción con objetos reales de otras clases asumiendo que esos objetos funcionan correctamente.

## Tests aislados

Los tests aislados sólo permiten que los colaboradores del SUT, es decir, los objetos necesarios para probar el SUT sean test doubles. Los test doubles son réplicas de objetos reales con atributos inventados que sirven para verificar que no hay ningún problema relacionado con los colaboradores del SUT cuando se está probando un trozo de código.

# Testing de un framework

Debido a que un framework ya proporciona al desarrollador ciertas funcionalidades para ayudarlo en su trabajo, se entiende que el código perteneciente al framework puede considerarse “código trivial” en el sentido de que no debe ser testeado (se asume que el código que proporciona el framework funciona).

Sin embargo, debido a las funcionalidades añadidas que proporciona un framework como puede ser *SpringBoot*, no es posible realizar tests de una forma convencional.

Normalmente los frameworks proporcionan una forma de ayudar a los desarrolladores al hacer testing. En concreto *SpringBoot* proporciona las anotaciones de *@DataJpaTest.*

# Test doubles

Los test doubles ahorran la preocupación de utilizar objetos con valores válidos cuando estos no afectan al SUT. Existen varios tipos de test doubles:

* Stub: No poseen logica interna, devuelven respuestas predeterminadas.
* Mock: Esperan ser llamados y son útiles para probar interacciones entre objetos.
* Fakes: Son implementaciones livianas de la API que se comportan como la real y que han sido implementadas previamente por el equipo de desarrollo.
* Dummy: Usados para llenar parámetros, pero no son un objeto real.

# Cobertura de un test

Para asegurar que se cubren todos los casos posibles en nuestras pruebas es conveniente seguir las siguientes estrategias:

* Caso positivo/negativo: En los bloques de sentencia “if” se deben probar todos los casos posibles. Es decir, es necesario probar el caso positivo de la sentencia dentro del “if” y el caso negativo. Todas las sentencias de tipo “if” o “else” del bloque también tiene que ser probadas usando esta lógica.
* Ejecutar todas las sentencias: Hay que asegurar que cada línea de código se ejecuta al menos una vez.
* Ninguna/Una/Varias iteraciones: En los bucles “for” y “while” es necesario generar varios casos de prueba. El primer caso por probar es en el que no se recorre ni una vez el bucle. En el segundo caso hay que ejecutar una iteración del bucle. Por último, hay que probar a realizar varias iteraciones del bucle.
* Testear posibles rangos: Cuando probamos un rango de posibles valores hay que comprobar siete posibles valores o cinco, según el caso. Dos casos de prueba en los extremos del intervalo, dos en valores válidos cercanos a los extremos, uno entre el rango de valores, y dos fuera del rango de valores si es posible.
* Testear posibles valores: Se deben probar los casos en los que las funciones esperan un valor, teniendo en cuenta que dicho valor puede ser nulo o estar vacío.

# Buenas prácticas en el testing

A la hora de llevar a cabo el testing es conveniente seguir un conjunto de buenas prácticas. Algunas de estas prácticas son:

* Test paramétricos: Se usan para comprobar el comportamiento de distintos valores para el mismo test. Este tipo de test reduce la cantidad de código duplicado en los test.
* Fluent Assertion: Cuando se introducen asserts en los test es recomendable primero poner el valor a comprobar y segundo el valor con el que hay que compararlo. Esto facilita la lectura al desarrollador.
* Test unitarios enfocados: Es conveniente que los test unitarios se encarguen de probar una sola cosa. De esta forma se puede ver de forma más clara que escenario de los test ha fallado. Además, permite que el resto de test se ejecuten si uno de ellos falla.
* Efecto causa cercanos: Es recomendable mantener cerca los datos del test de lo que se prueba. Así facilita el mantenimiento y es más legible.
* Principio DRY: Son las siglas de “Don’t Repeat Yourself”. Este principio recomienda no duplicar código. Aunque en los test es más difícil seguir este principio debido al factor humano que necesita a veces repetir código para mejorar la legibilidad del test.
* Principio DAMP: Son las siglas de “Desciptive And Meaningful Phrases”. Este principio permite introducir redundancia en el código en post de una mejor legibilidad.

# Conclusión

El testing en un WIS es una actividad fundamental para garantizar que el sistema cumple con los requisitos del cliente y funciona sin errores inesperados. A lo largo de este documento, se han descrito los distintos tipos de pruebas, estrategias de cobertura y buenas prácticas que permiten mejorar los test de nuestro sistema.

El uso adecuado de técnicas como los test unitarios, de integración y End to End, junto con la aplicación de test doubles, asegura un balance eficiente entre cobertura y rendimiento.

En definitiva, el testing no solo es una fase del desarrollo, sino una práctica continua que contribuye a la fiabilidad, mantenibilidad y éxito del sistema final.

# Bibliografía

* 06 – Anexxes.docx
* Diapositivas de la asignatura DP1 (Temas 7 y 8)