**Beneficios de los tests y tipos**

**Beneficios**

* Comprobar los requerimientos de nuestra aplicación.
* Documentación y ejemplos de nuestras clases.
* Mediante Test Driven Development (TDD) nos puede ayudar en el diseño de clases.
* Confianza al desarrollar.
* Confianza para refactorizar nuestro código.
* Es una habilidad que se solicita cada vez más en el mercado.

Existen test automáticos y manuales, los automáticos van a requerir tiempo de desarrollo y algunas veces no serán tan viables, pero de ser posible siempre trata de hacer test automáticos ya que:

* Son más rápidos.
* Son más fiables.
* Son incrementales.
* Menos tiempo de desarrollo

**Tipos de test**

* **Unitario**: realizan pruebas a una función o clase muy concreta de nuestro proyecto.
* **Integración**: prueban cómo se conectan diferentes componentes de nuestro proyecto.
* **Funcionales**: prueban una funcionalidad de nuestro proyecto, pueden involucrarse varias clases.
* **Inicio a fin**: prueba todo un proceso del proyecto.
* **Estrés**: útil para probar si nuestra aplicación puede soportar grandes cantidades de procesos y peticiones a la vez.

**Test unitario con JUnit**

La función assertEquals de JUnit se encarga de comprobar que dos valores sean iguales, en este caso comprobar si nuestra función repeat retorna el valor esperado.

Debemos indicarle a JUnit mediante @Test que función va a realizar una prueba.

**Organización de tests con JUnit**

La forma correcta de separar nuestras pruebas es realizar cada una en su propia función, además, el nombre de la función debe describir que estamos probando.

Para indicarle a JUnit que esperamos una excepción lo debemos hacer de la siguiente forma:

@Test(expected = IllegalArgumentException.class)

**Test con Mockito: para simular un dado**

[Mockito](https://site.mockito.org/) nos va a servir para simular clases mientras probamos.

Para instanciar un mock debemos utilizar la función Mockito.mock() e indicarle como parámetro la clase que va a simular.

Las funciones assertFalse y assertTrue tal como su nombre lo indican, sirven para comprobar si un valor es igual a false o true respectivamente.

**Análisis de los tests y mejoras**

Nuestros test siguen un mismo proceso:

1. Se preparan los objetos que vamos a probar.
2. Llamamos al método que estamos probando.
3. Comprobamos los resultados.

Podemos reducir la cantidad de código moviendo las partes comunes de preparación a una función que se ejecute antes de cada prueba.

Con @Before le indicamos a JUnit la función que debe ejecutar antes de cada prueba.

**TDD: definición, beneficios, ciclo y reglas**

El Test Driven Development (**TDD**) o desarrollo guiado por test, creado por Kent Beck, consiste en escribir primero los test antes que las clases permitiéndote ver si el diseño de una clase es la adecuada.

**El ciclo del TDD**

* Red: escribe un test que falle.
* Green: escribe el código necesario para que pase el test.
* Refactor: mejora el código.

**Reglas**

1. Sólo escribirás código de test hasta que falle.
2. Sólo escribirás código de producción para pasar el test.
3. No escribirás más código de producción del necesario.

Puedes combinar las reglas del TDD con su ciclo tal como hizo el profesor:

1. Red: Escribirás el mínimo de código test que falle.
2. Green: Escribirás el mínimo de código de producción que pase el test.
3. Refactor: sólo cuando los tests estén pasando.

**Organización de una aplicación**

Por lo general una aplicación se divide en:

* **Interfaz**: Se encarga de la comunicación con el exterior o un usuario.
* **Negocio**: Es la lógica de nuestra aplicación.
* **Datos**: Se encarga de guardar los datos de nuestra aplicación.

Cada capa se puede comunicar con otra, pero no conoce los detalles de implementación.

**Test a partir de requerimiento**

Lo ideal es tener los requerimientos del proyecto bien especificados, habrá veces donde el programador deba concretarlos. Una vez establecidos los requerimientos empezaremos a escribir los test.

Existen diferentes escenarios para nuestros test:

* Típicos (Caminos felices)
* Extremos
* Incorrectos
* No previstos (Bugs)