UTN F r a
Avellaneda

Programación II

Clase 08

Programación Orientada a Objetos

Temario

Testing

- Definición
- Objetivo
- Tipos de Testing

Unit Testing

- ¿Qué es?
- FIRST
- Estructura
- Naming

Test Doubles

- Problemática
- Solución
- Tipos

Test-Driven Development

- Definición
- Metodología
- Ventajas

01.

Testing

Testing

Definición

PATRONES DE DISEÑO

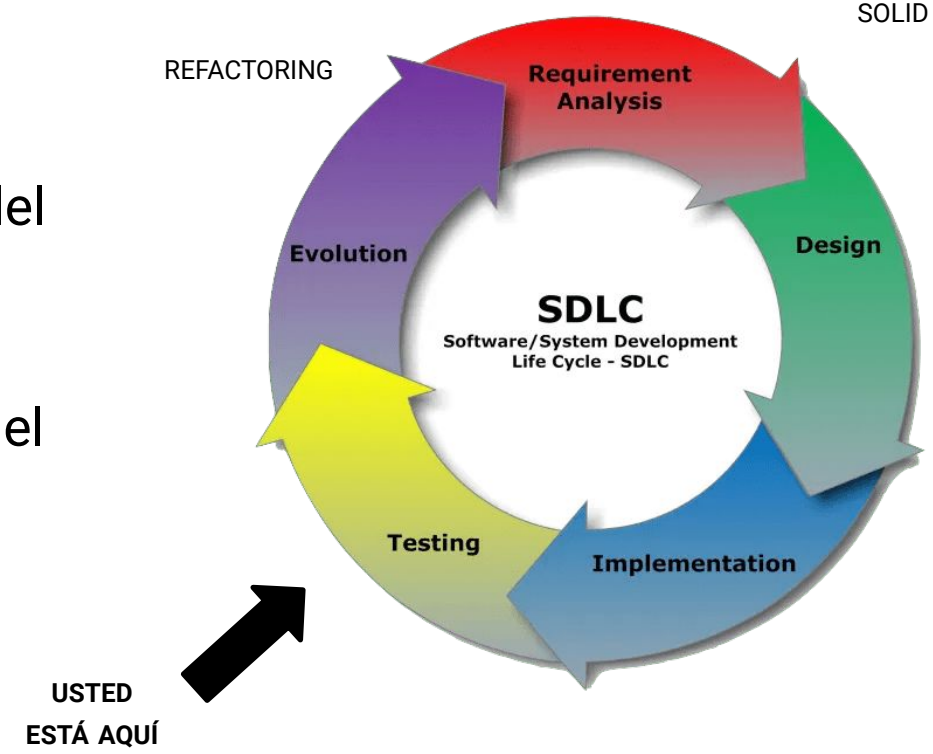
PRINCIPIOS SOLID

REFACTORING

- **Práctica**

  Técnica o metodología que forma parte del ciclo de vida del desarrollo de software.

  Mejora la calidad, eficiencia y velocidad del desarrollo.



Testing

Objetivo

- **Verificación del Desarrollo**

  Comprobar que nuestra aplicación realiza las acciones requeridas de manera correcta.

  ¿EL PRODUCTO FUNCIONA CORRECTAMENTE?

- **Validación del Producto**

  Comprobar que nuestra aplicación cumple con los requerimientos solicitados por el cliente.

  ¿EL PRODUCTO ADECUADO?

- **Integridad**

  Comprobar que los requerimientos anteriores se siguen cumpliendo luego de las sucesivas evoluciones.

Testing

Tipos

- **Funcionales**

DEVELOPERS

Pruebas unitarias (unit).
Pruebas de integración.
Pruebas de sistema.
Pruebas de aceptación.

- **No Funcionales**

Pruebas de rendimiento.
Pruebas de estrés.
Pruebas de usabilidad.
Pruebas de seguridad.

02.

Unit Testing

Unit Testing

¿Qué es?

- **Método**

  Porción de código (función o método) que verifica acciones o resultados de un método (unidad) con un resultado esperado.

  Si no coincide, el test falla.

- **Unidad**

  Nos referimos a unidad como *unidad de trabajo*, es decir, a la tarea que se activa cuando se invoca a un método público.

Unit Testing

F.I.R.S.T

- **Fast**

En ejecución y legibilidad.

La ejecución de los test no debe durar muchos segundos.
Si tardan, inevitablemente se evita ejecutarlos dando lugar a bugs y errores de diseño.

Si no son fáciles de leer, los errores serán más difíciles de encontrar y corregir. Además, son susceptibles a refactoring, lo cual también se torna más complicado.

Unit Testing

F.I.R.S.T

- **Independent**

  Los tests no deben depender unos de otros. Deben estar aislados.

  Pueden ejecutarse de manera concurrente: al mismo tiempo y sin orden.

Unit Testing

UTNFra Avellaneda

F.I.R.S.T

- **Repeatable**

  Replicable en cualquier momento.

  Pueden ejecutarse en cualquier plataforma (Linux, Windows), en cualquier entorno (Dev, QA), con o sin conectividad (Internet) y, principalmente, en cualquier momento. Siempre arrojan el mismo resultado.

Unit Testing

F.I.R.S.T

- **Self-Validating**

  Pasa o no pasa. Automático.
  No debe haber necesidad de revisar logs para ver el resultado.

- **Timely**

  Los tests deben estar escritos antes de que la app esté en producción.

Unit Testing

Estructura

- **Arrange**

  Preparar el contexto para la ejecución.
  Instanciar objetos, popular una lista, etc.

- **Act**

  Realizar la acción.
  Invocar al método.

- **Assert**

  Verificar resultados.
  Valor de retorno o modificación de atributos o estados.

Unit Testing

Naming

- **Convenciones para nombrar tests**

Aportan a la legibilidad.

GetNombre_NuevoAlumno_RetornaNombre

NOMBRE DE MÉTODO | **CONTEXTO EN EL QUE SE INVOCA** | **RESULTADO ESPERADO**

03.

Test Doubles

UTN F r a Avellaneda

Problemática

- **Colaboración**

En POO, los objetos colaboran (interactúan) entre sí para poder crear un sistema que resuelve una problemática, es decir, por lo que en los métodos que queremos probar con seguridad existen dependencias.

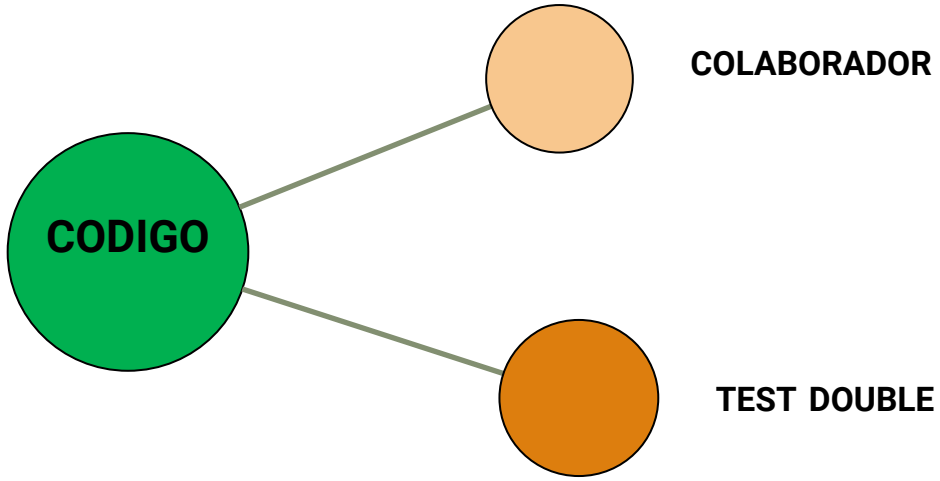
¿Cómo podemos entonces realizar test *unitarios*?

Test Doubles

Solución

- **Test Doubles**

Son objetos *fakes* o *dummies* en el cual **podemos establecer** el comportamiento que tienen, sin necesidad de que el test ejecute la implementación real.



Test Doubles

UTNFra Avellaneda

Ventajas

- **Aislación**

El bloque de código que queremos testear ahora está completa aislado del resto del código y podemos ejecutarlo como una unidad verdadera.

- **Velocidad**

Al reemplazar las interacciones con código real, la ejecución de los tests se resuelve de manera más rápida.

Test Doubles

Ventajas

- **Determinista**

Nos permite introducir el comportamiento de un objeto (colaborador), incluso antes de implementarlo.

Se obtiene completo control del contexto, evitando trabajar con datos aleatorios y, a la vez, establecer condiciones especiales.

UTN F r a Avellaneda

Tipos de Test Doubles

- **Stub**

Un *stub* no hace nada. Se trata de una implementación sencilla de un objeto con el que deseamos interactuar, pero del que no nos importa qué hace.

Podemos instanciarlo varias veces y cada vez su comportamiento puede ser distinto.

UTN F r a Avellaneda

Tipos de Test Doubles

- **Fake**

Más elaborado que el *stub*. Permite una implementación más completa pero que sigue siendo una versión light de la implementación real.

Un ejemplo común es cuando trabajamos con base de datos. No queremos realizar *queries* reales sino retornar datos concretos.

UTNFra Avellaneda

Tipos de Test Doubles

- **Spy**

Se utiliza en casos dónde la acción a testear no retorna ningún tipo de dato, pero, al existir una colaboración, permite interceptar las interacciones con este objeto.

Por ejemplo, determinar si uno de sus métodos fue llamado y con qué parámetros.

UTNFra Avellaneda

Tipos de Test Doubles

- **Mock**

Es una combinación entre *spy* y *stub*. Permite configurar el comportamiento de uno o varios métodos del objeto estableciendo el contexto para cada respuesta.

Son los más utilizados por su maleabilidad. Estos utilizaremos en los ejemplos.

04.

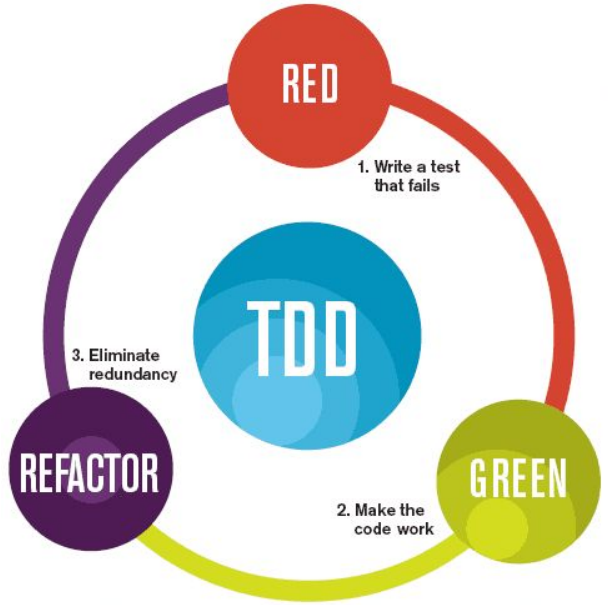
Test-Driven Development

Test-Driven Development

Definición

- **¿En qué consiste?**

La idea es, tomando un requerimiento, crear un test.
Claramente, va a **fallar** ya que no está implementado.
Desarrollar el código para que el test **pase**.
Refactorizar.



Test-Driven Development

Ventajas

- **Lo justo y necesario**

Permite desarrollar exactamente lo requerido sin caer desarrollos deficientes (que no cumplen lo requerido) o sobrediseño (agregar más de lo pedido por las dudas).

Además, permite identificar vicios en los requerimientos.

- **Alta cobertura**

No (casi) código sin testear, lo que lo hace más resistente a bugs.