

Capacitación Pruebas Unitarias (xUnit)

11 Noviembre 2022 – Sesión 4

**FUTURE
AT HEART**

Contenido



Modulo I – Conceptos Básicos

- ¿Qué es una prueba unitaria?
- Otros niveles de prueba
- Conociendo las pruebas unitarias
- Técnicas de diseño de casos de prueba
- Principios FIRST

Modulo II – Test Driven Development

- Definición
- Desarrollo Ágil – Características
- Ciclo de desarrollo TDD
- Como escribir código que se pueda probar
- Cobertura de código
- Ventajas/Desventajas

Modulo III – xUnit

- ¿Qué es xUnit?
- ¿Por qué xUnit?
- ¿Cómo funciona xUnit?
- Conceptos básicos
- Escenarios con xUnit
- Patrones de nombramiento de los escenarios
- Simulaciones
- Aserciones

Modulo IV – Taller

- Preparación del Entorno
- Ejemplo práctico
- Creando casos de prueba
- Ejecución de casos
- Mutation Testing
- Conclusiones/Recomendaciones

En capítulos anteriores...



Libros > Computadoras y Tecnología > Programación



Escuchar



Ver esta imagen

Sigue a los autores



Robert C. Martin

Seguir

Clean Code: A Handbook of Agile Software Craftsmanship 1st Edición

de Robert C. Martin (Author)

★★★★☆ 4,703 calificaciones 4.4 en Goodreads 19,245 calificaciones

Parte de: Robert C. Martin (14 libros)

Ver todos los formatos y ediciones

Kindle
US\$28.99

Leer con nuestra **Aplicación gratuita**

Pasta blanda
US\$42.10 - US\$44.99

7 Usado de US\$39.31
2 Nuevo de US\$44.99

Even bad code can function. But if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have to be that way.

Noted software expert Robert C. Martin, presents a revolutionary paradigm with *Clean Code: A Handbook of Agile Software Craftsmanship*. Martin, who has helped bring agile principles

Leer más

Reportar información de producto incorrecta

ISBN-10



9780132350884

ISBN-13



978-0132350884

Edición



1er

Editorial



Pearson



2008 - 2022

14 Años

Estructura de las pruebas unitarias

NTT DATA

AAA
INVEST



Por norma general, los unit test deberían seguir una estructura AAA, Este patrón fue creado por BILL WAKE (2001) y esta compuesto por:

- **Arrange** (Organizar): En esta parte de la prueba, debes establecer las condiciones iniciales para poder realizarla, así como el resultado que esperas obtener. Y esto significa por ejemplo, declarar las variables y crear las instancias de los objetos.
- **Act** (Accionar): Es la parte de ejecución, tanto del fragmento de código de la prueba, como del fragmento de código a testear.
- **Assert** (Comprobar o certificar): Por último, se realiza la comprobación para verificar que el resultado obtenido, coincide con el esperado.

```
[TestMethod]
public void TestMethod()
{
    //Arrange test
    testClass objtest = new testClass();
    Boolean result;

    //Act test
    result = objtest.testFunction();

    //Assert test
    Assert.AreEqual(true, result);
}
```

Estructura de las pruebas unitarias

```
public class Suma {  
    public int sumar(int numeroUno, int numeroDos){  
        return numeroUno + numeroDos;  
    }  
}
```

@Test

```
public void sumarDosNumerosPositivos(){
```

//Arrange

```
Suma suma = new Suma();
```

```
int numeroUno = 2;
```

```
int numeroDos = 3;
```

```
int resultadoObtenido;
```

//Act

```
resultadoObtenido = suma.sumar(numeroUno,numeroDos);
```

//Assert

```
Assert.assertEquals( expected: 5, resultadoObtenido);
```

1

2

3

Beneficios:

- Estándar (Mantenimiento)
- Se elimina la duplicidad
- Separación clara

1[

Ingredientes

- ☐ 250 gramos de harina
- ☐ 150 gramos de mantequilla
- ☐ 150 gramos de azúcar
- ☐ 5 huevos enteros
- ☐ 1 cucharada copeteada de polvo para hornear
- ☐ 1 cucharada de vainilla

Preparación

✓ Paso 1

Precalienta el horno a 175° centígrados y engrasa y enharina un molde para hornear.

✓ Paso 2

Bate la mantequilla con el azúcar hasta que acrete y, sin dejar de batir, agrega

✓ Paso 3

Cierne la harina con el polvo para hornear en incorpóralo a la mezcla anterior y añade la vainilla.

Resultado esperado



2[

3[

Arrange

Act

Assert

Fast (rápido)

Independent (independiente)

Repeatable (repetible)

Self-validating (auto evaluable)

Timely (oportuno)

Principios FIRST - F

Fast (rápido)

Los test unitarios deben completar su ejecución lo más rápido posible. Al ser pruebas que se realizan (o deberían realizarse) sobre fragmentos pequeños de código, deben finalizar su ejecución lo antes posible.



Independent (independiente)

En las pruebas unitarias, el objeto de prueba no debe depender de otra unidad de código. Esto no significa que no requiera de datos para poder ejecutar ese código, pero esto se realizará a través de **mocks o stubs**, que vienen a ser «objetos falsos» creados específicamente para realizar las pruebas. Pero, en ningún caso, utilizaremos otras partes del código que no forme parte del unit test.

Dummy : Objetos ficticios que se pasan al método pero no se usan. Por lo general, solo se usan para rellenar listas de parámetros.

Fake : Objetos falsos que toman un atajo para cumplir suprir un valor de entrada o componente. Ej: Base de datos en memoria.

Stubs : Respuestas enlatadas para un solo contexto de prueba. Ej: Un parámetro constante (Cedula, Tipo)

Spies : Proporcionan información específica reaccionando de acuerdo a como fueron llamados. Ej: Ingreso una cedula y me retorna # de líneas

Mocks : Objetos pre programados complejos (Subrutinas).

Cada prueba unitaria debe ser independiente de otra

- Se pueden ejecutar las pruebas en diferente orden
- El resultado no se ve alterado por el orden de ejecución de la prueba
- Su nombre lo dice prueba unitaria (No de integración)

Repeatable (repetible)

Las pruebas unitarias deben poder repetirse. De hecho, lo ideal es repetirla muchas veces. Por ejemplo, se deberían ejecutar las pruebas cada vez que suba un nuevo código al repositorio.

- En mi computadora si funciona
- Si pero no le vamos a dar tu computadora al cliente



Repetible independiente del

- Servidor
- Computador
- Entorno
- No depende de configuración de usuario
- No depende de configuración de las herramientas
- Me deben dar el mismo resultado en cualquier entorno

Self-validating (auto evaluable)

Los unit test deben mostrar de forma clara, y sin que sea necesario tu intervención, el resultado de la prueba. Gracias a la parte de «arrange», donde estableces los resultados a obtener, la parte de «assert» puede responder si la prueba ha sido satisfactoria o no.



Timely (oportuno)

Los test unitarios deben realizarse lo antes posible, el código no debe llegar a producción sin haber escrito y superado las pruebas unitarias. Incluso, si es posible, deberían desarrollarse antes que el código (Test Driven Development)



FAST SERVICE

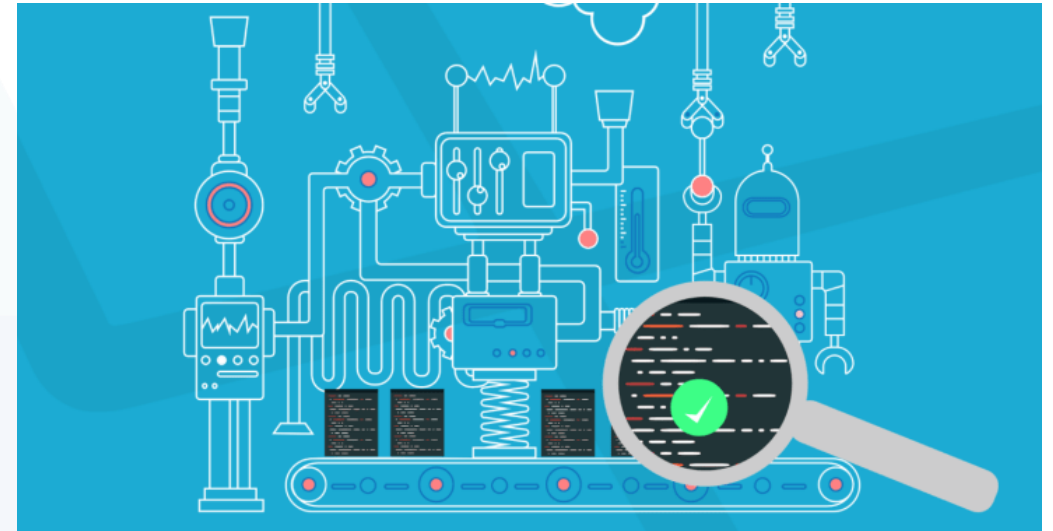
Cuando hacer o no pruebas unitarias

Se hacen para:

- Menos errores en producción
- Resolución mas rápida de errores
- Comprensión del código
- Sencillez de integración
- Mejoras en la estructura del código
- Localización de errores
- Reducción de costes

Cuando no es posible:

- El código cambia mucho
- El periodo de vida del producto es muy corto
- Componentes de menor importancia dentro del código con tiempos de entrega ajustados




TDD y Pruebas Unitarias

9 -Verdadero o falso

¿Xunit, Juniy y otros Fw de pruebas unitarias son lo mismo que TDD?

9 de 9



☒ True

☐ False

☐ Sin respuesta

✗	4
✓	10
✗	1

⌚

Límite de tiempo de 20 s

Respuestas correctas

67 %

Tiempo promedio de res...

9.96s

Jugadores respondieron

14 of 15

¿Que son las pruebas unitarias?

¿Que es TDD?

¿Que es BDD?



NTT DATA

GRACIAS

**FUTURE
AT HEART**