Telefónica

NTT DATA

Capacitación Pruebas Unitarias (xUnit)

23 Noviembre 2022 – Sesión 7

FUTURE AT HEART

Contenido

Modulo I – Conceptos Básicos

- ¿Qué es una prueba unitaria?
- Otros niveles de prueba
- Conociendo las pruebas unitarias
- Tecnicas 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

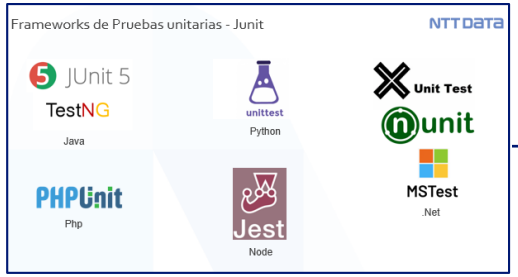
Modulo III – xUnit

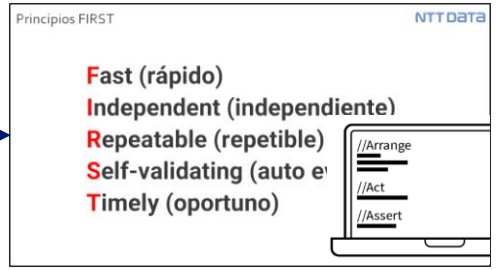
- ¿Qué es xUnit?
- ¿Por qué xUnit?
- ¿Cómo funciona xUnit?
- Conceptos básicos
- Escenarios con xUnit
- Patrones de nombramiento de los escenarios
- Simulaciones
- Aserciones

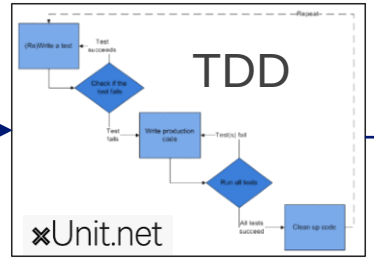
Modulo IV – Taller

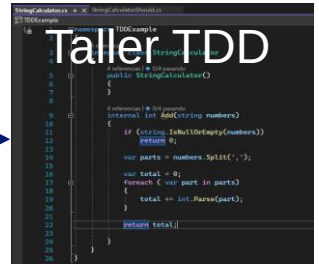
- Preparación del Entorno
- Ejemplo práctico
- Creando casos de prueba
- Ejecución de casos
- Mutation Testing
- Conclusiones/Recomendaciones

En capítulos anteriores...









https://github.com/oacarrillo/curso-xUnit

Como escribir código que se pueda probar

NTT DATA

1. **Nombres con sentido:** Nombra tus variables, constantes y funciones con un sentido holístico. Es decir, deben indicar por qué existe, qué hace y cómo se usa.

```
int p; // Pago en dólares
```

```
int paymentInUSD;
```

2. **Nombres que se puedan pronunciar:** Es altamente aceptado, sugerido y hasta obligado el uso del inglés para nombrar las variables, constantes y funciones. Esto es un estándar de la industria. Ahora, aunque tu no sepas inglés, los nombres deben poderse pronunciar. Te daré un ejemplo más claro:

```
date genymdhms;
```

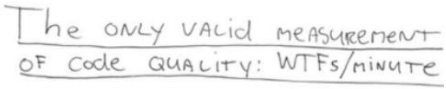
3. **Nombres que se puedan buscar :** Si nombramos nuestras variables con una sola letra, será muy difícil hacer un 'Ctrl+f' y buscar. En inglés la letra más usada es la 'e' y en español la 'a', así que imagínense utilizar una sola letra y buscarla en el código. Verán todo el IDE o editor de código resaltando todas las letras.

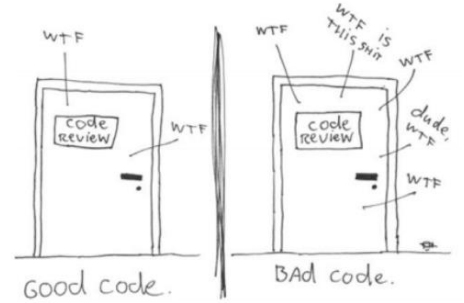
```
int a;
```

```
int aa;
```

4. **Uso de abreviaciones:** El libro recomienda no usar abreviaciones. Sin embargo, existen algunas abreviaciones que son tan comunes que tienen total sentido. Por ejemplo: 'source' podríamos cambiarlo por 'src', 'image' por 'img', entre otros. ¿Por qué? Porque 'src', 'img' y otras abreviaciones son ampliamente utilizadas, ejemplo:

```
<img src="path" alt="ejemplo de abreviaciones">
```





A representation i just found reading "Clean Code" by Robert C.Martin. It made my day.

https://ed.team/blog/11-consejos-para-escribir-codigo-limpio
https://pics.me.me/a-representation-i-just-found-reading-clean-code-by-robert-71901439.png

Como escribir buen código y que este se pueda probar

5. **Variables de una sola letra:** Ya dijimos que no deben usar variables de una sola letra. Pero... como toda regla, existen sus excepciones:

```
for (int i = 0; i < 10; i++) {
```

```
public String sayHello(n string) {
        return "Hello " + n
}
```

```
public String sayHello(name string) {
        return "Hello " + name
}
```

6. **Las funciones deben hacer una sola cosa:** Es normal que mientras programamos, vamos escribiendo todo lo que tiene que hacer nuestro software. Pero en ocasiones las funciones (o métodos) se vuelven gigantes. Cuando lo lees, vas revisando que hay cosas que deberían estar en otras funciones (o métodos) más pequeños que serán llamados por tu función principal, así que podrás utilizar el refrán 'Divide y vencerás'.

```
func burbleSort(numbers []int) {
        for i:=1; i < len(numbers); i++ {
                for j:=0; j < len(numbers)-1; j++ {
                        if numbers[i] > numbers[j] {
                                numbers[i], numbers[j] = numbers[j], numbers[i]
                        }
                }
        }
}
```

```
func burbleSort(numbers []int) []int {
        for i:=1; i < len(numbers); i++ {
                exchange(numbers, i)
        }
}

func exchange(numbers []int, i int) {
        for j:=0; j < len(numbers)-1; j++ {
                if numbers[i] > numbers[j] {
                        numbers[i], numbers[j] = numbers[j], numbers[i]
                }
        }
}
```

https://ed.team/blog/11-consejos-para-escribir-codigo-limpio

Como escribir buen código y que este se pueda probar

7. **Escribir código es como escribir un libro:** Cuando escribas código, debes pensar en que al leerlo debes poder hacerlo como si estuvieras leyendo un libro. Con un orden, sabiendo que cada capítulo está enfocado a lograr un objetivo pequeño y al leer todo el libro habrás entendido el objetivo general.
8. **Las funciones no deben tener efectos secundarios:** Las funciones deben hacer una sola cosa y la deben hacer bien.

```
// BAD
func isValidPayment(payment, coursePrice int) bool {
        if payment == coursePrice {
                createInvoice()
                return true
        }

        return false
}
```

```
// GOOD
func isValidPayment(payment, coursePrice int) bool {
        return payment == coursePrice
}
```

9. **Comentarios en el código:** El libro sugiere que no uses comentarios. Pero, las review que he visto del libro han sido extremistas al decir que 'jamás' se deben usar. Eso no es cierto, lo que dice el libro es que los comentarios de deben usar siempre y cuando agreguen valor al código.

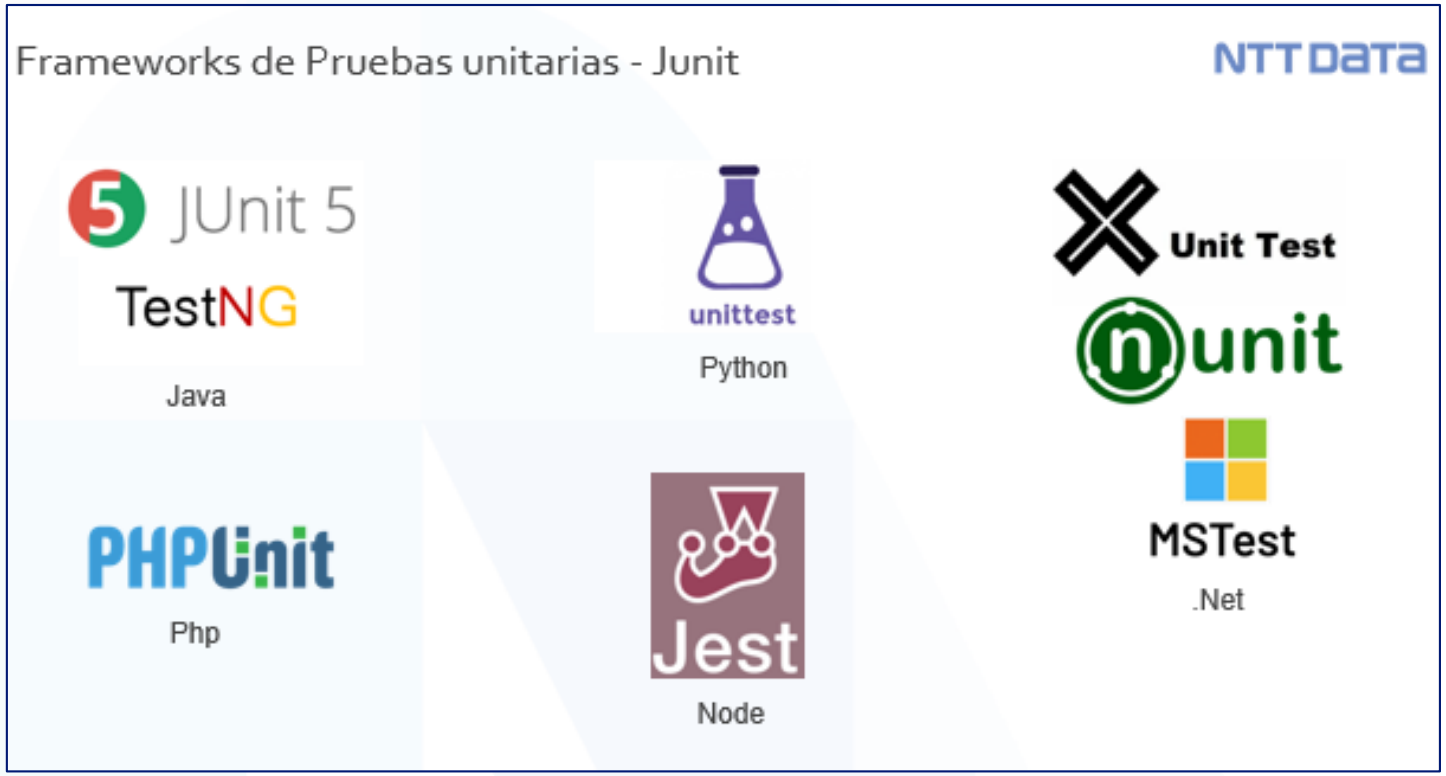
```
var dayOfMonth int // día del mes
func validateUser(u user) bool { // Valida el usuario
return value // retorna el valor
// Cambio de código hecho por Fulano de tal el 08/08/2008
```

```
// calculateOneByOne calcula los registros de manera individual.
// Debe usarse con cuidado ya que el rendimiento de esta función
// puede afectar seriamente el proceso de facturación del mes
func calculateOneByOne() {
```

https://ed.team/blog/11-consejos-para-escribir-codigo-limpio

Como escribir buen código y que este se pueda probar

10. **Hacer testing:** Hay muchas razones para hacer testing de tu código: Refactorización, mejora de rendimiento, corrección de bugs, etc. Pero para el tema de código limpio, hacer testing te permite dividir tu aplicación de forma correcta, haciendo que las funciones/métodos cumplan con una sola función, te ayuda a desacoplar tus clases/estructuras de acoplamientos innecesarios, etc.



https://ed.team/blog/11-consejos-para-escribir-codigo-limpio

Junit - Profundización

JUnit es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

JUnit es un conjunto de clases (*framework*) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

https://junit.org/junit5/docs/current/user-guide/


NTT DATA
GRACIAS
FUTURE
AT HEART