Telefónica

NTT DATA

Capacitación Pruebas Unitarias (xUnit)

6 de Diciembre 2022 – Sesión 11

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

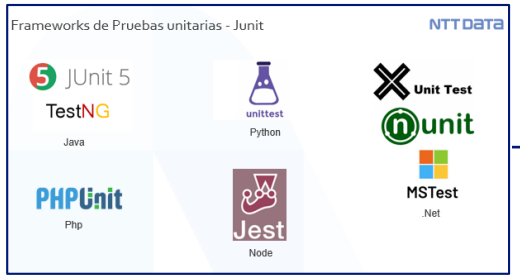
- Preparación del Entorno
- Ejemplo práctico
- Creando casos de prueba
- Ejecución de casos
- Mocks
- Mutation Testing
- Conclusiones/Recomendaciones

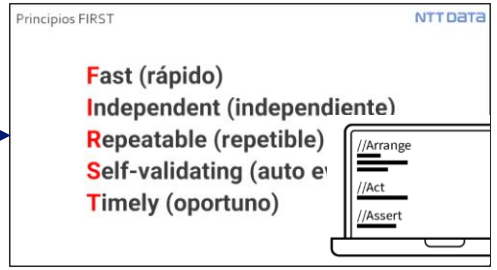
Bonus

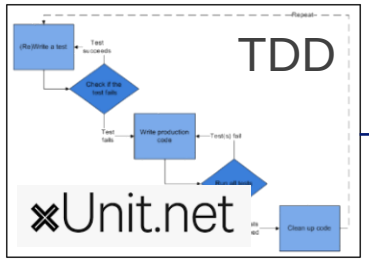
- Pruebas API
- Junit + Selenium
- Jest

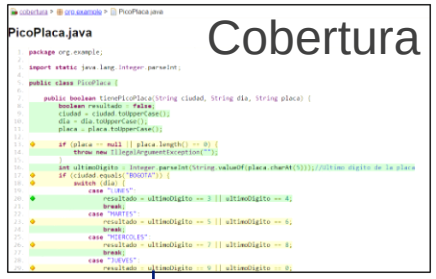
En capítulos anteriores...

NTT DATA









Mocks

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

Mutation testing

¿Que es? : Es una técnica para medir la calidad de los tests a largo plazo. En las pruebas de mutación, mutamos (cambiamos) ciertas declaraciones en el código fuente y comprobamos si los casos de prueba son capaces de encontrar esos errores introducidos. Es un tipo de pruebas de caja blanca que se utiliza principalmente para las pruebas unitarias. Los cambios en la aplicación «mutante» se mantienen muy pequeños, por lo que no afecta al objetivo general de la aplicación.

Objetivo: El objetivo de las pruebas de mutación es evaluar la calidad de los casos de prueba que deben ser lo suficientemente robustos como para fallar ante código mutante. Este método también es llamado estrategia de evaluación basada en fallos, ya que implica la creación de fallas en el programa.

https://josepablosarco.wordpress.com/2015/07/15/pruebas-de-mutacion-mutation-testing/

Mutation testing

Pasos para ejecutar un mutation testing :

- Corremos los tests sobre el Código fuente -> pruebas pasan
- Corremos los tests sobre el Código Mutante -> pruebas pasan (Mutante sobrevive!)
- Corremos los tests sobre el Código Mutante -> pruebas fallan (¡Mutante Muere!)

Como podemos ver, que el mutante sobreviva nos indica que cambiando el código fuente, los tests siguen pasando, ¡algo muy pero muy mal! En cambio, si corremos las pruebas sobre el código mutante y las mismas fallan, esto indica que las pruebas fueron capaces de detectar el cambio, ¡lo que es muy bueno!

PIT es una librería que permite realizar *mutation testing* en Java. Las operaciones de mutación que realiza en el código pueden ser en los condicionales, incrementos, invertir negativos, matemáticas, negar condicionales, cambiar los valores de retorno o eliminar llamadas a métodos sin retorno. Un ejemplo de mutaciones son realizar mutaciones en los límites de comparaciones, cambiando un < por un <= y comprobar si con operador mutado la mutación sobrevive pasando todos los teses.

Original	Mutación
<	<=
<=	<
>	>=
>=	>

https://josepablosarco.wordpress.com/2015/07/15/pruebas-de-mutacion-mutation-testing/

Mutation testing

Ventajas:

- Trae un nuevo tipo de errores a la atención de los desarrolladores.
- Es el método más poderoso para detectar defectos ocultos, que podrían ser imposible identificar mediante las técnicas de pruebas convencionales.
- Es un complemento a la medida de cobertura de las pruebas.
- Trae un buen nivel de detección de errores para las pruebas de unidad.
- Descubre ambigüedades en el código fuente.
- Se obtienen sistemas más fiables y estables.

Desventajas:

- Extremadamente costoso y consume mucho tiempo, ya que hay mucho código mutante que necesita ser generado (Por esto se dejo de usar luego de ser una practica muy famosa allá por 1971)
- Muy difícil de aplicar sin una herramienta de automatización (por su costo)
- Cada mutación tendrá el mismo número de casos de prueba que la de el programa original. Por lo tanto, puede necesitar ser probado contra el conjunto de pruebas original de un gran número de programas de mutantes.
- Implica cambios en el código fuente, por lo que no es en absoluto aplicable para las pruebas de caja negra.

https://josepablosarco.wordpress.com/2015/07/15/pruebas-de-mutacion-mutation-testing/

Jest

Jest es un marco de prueba de JavaScript encantador que se centra en la simplicidad. ¡Funciona con proyectos que usan: Babel , TypeScript , Node , React , Angular , Vue y más!

Jest es un marco de prueba de JavaScript creado sobre Jasmine y mantenido por Meta (anteriormente Facebook) . Fue diseñado y construido por Christoph Nakazawa con un enfoque en la simplicidad y soporte para grandes aplicaciones web . Funciona con proyectos que utilizan Babel, TypeScript, Node.js ,React ,Angular ,Vue.js y Svelte. Jest no requiere mucha configuración para los usuarios primerizos de un marco de prueba.


NTT DATA
GRACIAS
FUTURE
AT HEART