THE NINJ PROJECT

Microservices 101.TDD

Raimundo Alegría

THE NINJA PROJECT

El objetivo de este learning path es recorrer de una forma práctica algunos los aspectos de los aspectos más relevantes que influyen tener éxito a la hora de desarrollar una arquitectura de microservicios

THE NINJA PROJECT

Talleres previstos. Microservices 101

- Conceptos básicos
- ATDD
- TDD (Conceptos básicos)
- Containers. Docker
- Infraestructura. Kubernetes
- Continuous delivery
- Seguridad
- Bases de datos
- Event sourcing
- Arquitecturas evolutivas

THE NINJA PROJECT

¿Qué es?

THE NINJA PROJECT

Test driven development, es una forma de escribir software de manera evolutiva donde primero se escribe una prueba y después desarrolla únicamente el suficiente código para hacer que cumplir esa prueba.

THE NINJA PROJECT

Características

- Permite el diseño emergente, el refactor es imposible sin test unitarios.
- Es una es técnica clave para la integración continua o continuous delivery.
- Feedback rápido. Fail fast.

THE NINJA PROJECT

Características

- Mejoran la calidad y reducen los bugs.
- Hacen que el código sea más simple.
- La prueba es el primer cliente de la clase o función.
- Generan un conjunto de tests de regresión.

THE NINJA PROJECT

Características

- Crea una especificación detallada de la funcionalidad. La documenta usando ejemplos ejecutables.
  - Documenta el contrato del método explícitamente.
- Tener un set de pruebas permite al sistema cambiar más rápido.

THE NINJA PROJECT

Ciclo TDD

- Rojo: Crea un test unitario que falla.
- Verde: Escribe el mínimo código en producción que haga que pase la prueba.
- Refactor: Limpia el lío que has creado.

Make it work. Make it right. Make it fast.

THE NINJA PROJECT

Anatomía de un test

- Expectativas
  - Cuál es el resultado esperado de la prueba.
  - Assert per test.
  - Assert first.
- Experimentación
  - El experimento concreto que se va a realizar.

THE NINJA PROJECT

Anatomía de un test

- Fixture
  - Establece el escenario para la prueba, configura el sistema en el estado inicial para probarlo.
  - Fresh fixture. Cada prueba no comparte ningún dato con otras pruebas.
  - Test wars. Cuando dos o más pruebas comparten datos y estos no se limpian adecuadamente.

THE NINJA PROJECT

Demostración. Test simple

- Borrar la implementación actual del método. Hacer que compile devolviendo null.
- En java las pruebas se anotan con @Test.
- La pruebas normalmente empiezan por should…
  - Te hace pensar lo que la clase debería hacer.
  - void should_not_return_null … o shouldNotReturn...

THE NINJA PROJECT

Demostración. Test simple

- Decidir la característica más simple a implementar.
- Escribir el primer assert.
- Implementar la llamada al método que queremos probar
- Hacer que la clase compile, completar la fixture.
- Pasar las pruebas.

THE NINJA PROJECT

Demostración. Test simple

- Rojo. La prueba debe fallar
- Escribir el mínimo código para
- implementar el método.
- Pasar la prueba.
- Verde :-).
- Refactorizar.

THE NINJA PROJECT

Primer ejercicio

- Ir a: https://github.com/rai22474/microservices-101/wiki/TDD
- Ir a **ejercicio 1 TDD**.
- Seguir la instrucciones.

THE NINJA PROJECT

Primer ejercicio

- Hacer el resto de las pruebas del método.
  - Elegir otra prueba a realizar.
  - Repetir el proceso anterior de la demostración.
- 30 minutos

THE NINJA PROJECT

Tipos de test

Caja negra

- El test no sabe nada sobre la estructura interna de la clase o función.
- Únicamente sabe sobre las entradas y salidas.
- El test y la implementación no están acoplados.
- Fácil de refactorizar.
- Fácil de implementar en funcional puro o sin side effects.

THE NINJA PROJECT

Tipos de test

Caja blanca/transparente

- El test conoce la estructura interna de la clase o función.
- Hay un grado de acoplamiento entre el test y la implementación
- Se suelen usar para probar métodos que tienen side effects.
- Hace el refactor más complicado.

THE NINJA PROJECT

Mi estrategia de pruebas

Test first

El test se escribe antes que el código

- Ahorra mucho esfuerzo de debugging.
- El código se diseña para ser testable.
- Hace el diseño más simple.
- YAGNI.
- Se debe pensar a priori el mejor interfaz para el código.

THE NINJA PROJECT

Mi estrategia de pruebas

Prueba en aislamiento

Se prueba la clase de manera separada al resto de la aplicación.

- Cuando falla un test sabes exactamente dónde ha fallado.
- Permite razonar sobre cuales son las dependencias con otras clases o funciones.
- Se deben usar test doubles.

THE NINJA PROJECT

Mi estrategia de pruebas

Desarrollo de fuera a dentro

Primero se piensa en test de usuario en caja negra y se va desarrollando el sistema para cumplir esos tests.

- Hollywood principle.
- Inversion of Control.
- El sistema crece orgánicamente a partir de las necesidades del usuario.

THE NINJA PROJECT

Test doubles

Son objetos que sustituyen a las dependencias de la clase o función que está siendo probada.

- Stubs.
- Mocks.
- Fakes.
- Dummies

THE NINJA PROJECT

Ejemplos de test

- Test de un equals.
- Test en caja blanca cuando es necesario crear un objeto.
- Test de un extremo del sistema.
- Test de un método asíncrono en java con retries.
- Test de una lambda en java.

THE NINJA PROJECT

Lecciones aprendidas

- Se debe tratar el código de pruebas de pruebas de la misma forma que el código de producción:
  - Los tests deben ser independientes. Hay que evitar las guerras de test.
  - Es importante mantener los test DRY.
  - Intentar mantener los test simples.

THE NINJA PROJECT

Lecciones aprendidas

- Es preferible hacer test de caja negra que test de caja blanca o transparente.
- Doubles:
  - En los extremos del sistema es preferible usar fakes.
  - Es preferible usar stubs frente mocks.

THE NINJA PROJECT

Lecciones aprendidas

- Test asíncronos
  - Es conveniente evitar los sleeps en los test de funciones o método asíncronos.
  - Es preferible hacer reintentos con un tiempo.
- Los test no te hacen más inteligente.
- El set de test debe pasar en un periodo corto de tiempo para maximizar el feedback.

THE NINJA PROJECT

Muchas gracias

THE NINJA PROJECT

Referencias (Libros)

- xUnit Test Patterns: Refactoring Test Code. Gerard Meszaros.
- Test Driven Development. Kent Beck.
- Refactoring: Improving the Design of Existing Code . Martin Fowler.
- Clean code. Robert C. Martin.

THE NINJA PROJECT

Referencias (Links)

- http://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html
- http://junit.org/junit4/
- http://www.jmock.org/oopsla2004.pdf
- https://martinfowler.com/articles/mocksArentStubs.html