TDD in Spring Boot

Juan Jiménez
@jnsrjzgo

¿Qué es TDD?

Desarrollo guiado por pruebas de software, o Test-driven development popularizado por Kent Beck creador de XP.

Se comienza escribiendo los tests y nos ayuda a diseñar mejor nuestras clases.

Es un proceso que radica en la repetición de un corto ciclo de desarrollo.

TDD no es fácil de dominar, requiere tiempo y mucha práctica.

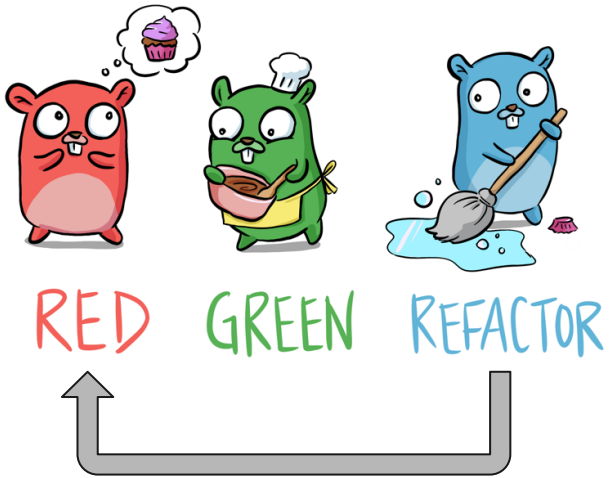
Changos!

Ciclo TDD red-green-refactor

RED: Escribir un test que no pase (Escribirás el mínimo código del **test** para que falle).

GREEN: Escribir el código necesario para pasar (Escribirás el mínimo código hasta que pase el **test**)

REFACTOR: Mejorar el código (Solamente cuando **los tests** estén pasando)

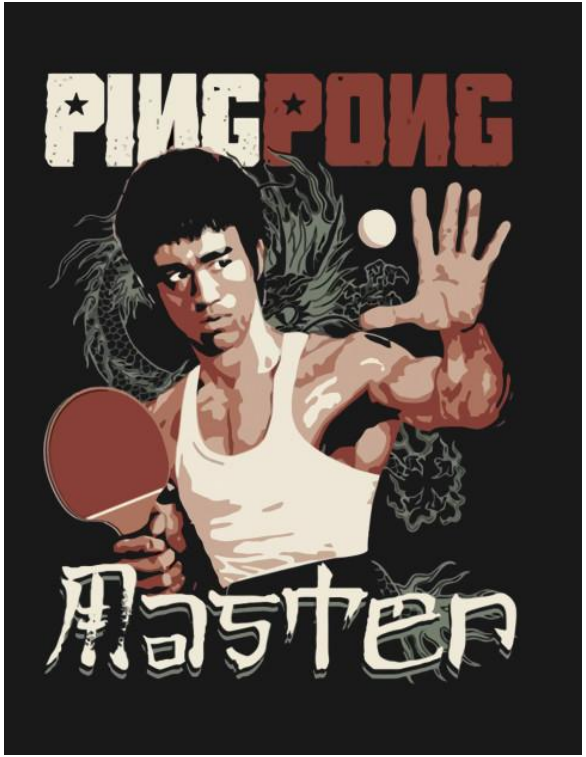


La velocidad es la clave!

Los expertos en TDD intentan no superar 1 minuto en generar su prueba y su implementación.

- Escribir una prueba pequeña y correr todas las pruebas (**ping**).
- Escribir la implementación y correr todas las pruebas(**pong**).
- Escribe otra prueba (**ping**).
- Escribe su implementación (**pong**).
- Refactorizar y confirma que todas las pruebas pasen (**score**).
- Y luego repítelo (**ping, pong, ping, pong, score, serve again**).

No trates de hacer el código perfecto desde la primera vez, mejor intenta que la pelota siga en juego hasta el momento justo para anotar (**refactor**). **El tiempo de armar las pruebas y la implementación debería ser medido en minutos, incluso segundos**.



Al final no se trata de probar

El objetivo principal de TDD tener un diseño de código testable con pruebas como un producto secundario muy útil.

No debugging

Se presume que usando TDD se reduce drásticamente el tiempo en depuraciones a las aplicaciones.

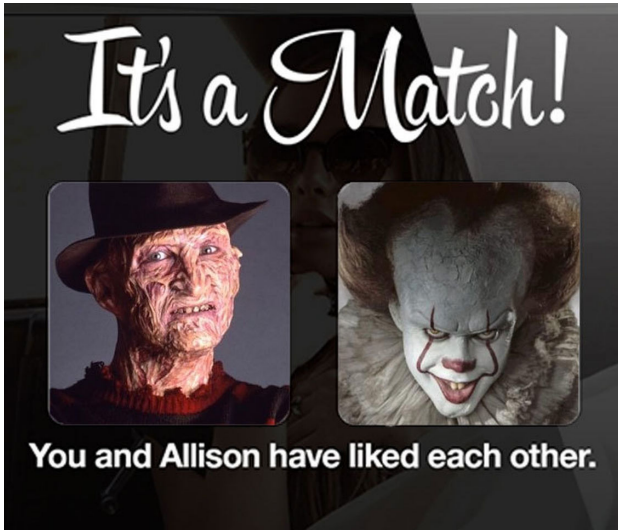
Cuando se tienen las pruebas escritas antes que el código y se tiene una alta cobertura es más fácil aislar el defecto identificando el código que no se cubrió por las pruebas.

Assertions

En muchos casos, es más fácil saber qué hace el código si revisamos las pruebas que lo implementan.

Hamcrest agrega varios matchers diseñados para realizar comparaciones que se pueden extender. De hecho JUnit soporta Hamcrest nativamente ya que se incluye en el core.

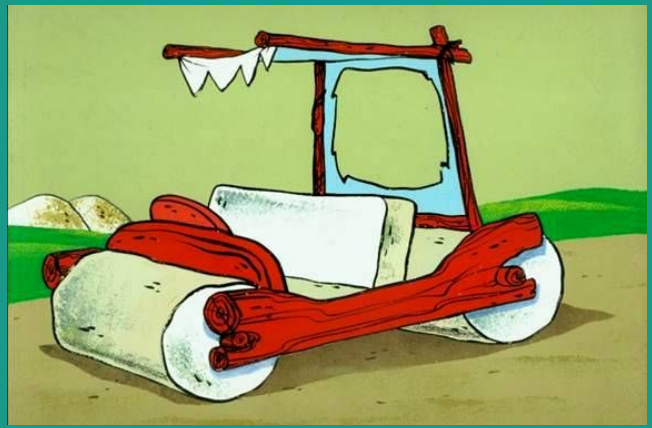
AssertJ las afirmaciones se pueden concatenar sin necesidad de tener dos afirmaciones separadas además de que **AssertJ** es más legible y fácil de entender.



Mocking

Simular dependencias externas nos permite incrementar la **velocidad** de pruebas de forma drástica. La batería **completa** de pruebas unitarias debería ser medida en **minutos**, incluso **segundos**.

Con o sin simulaciones, el código debería ser escrito de modo que podamos reemplazar fácilmente sus dependencias.

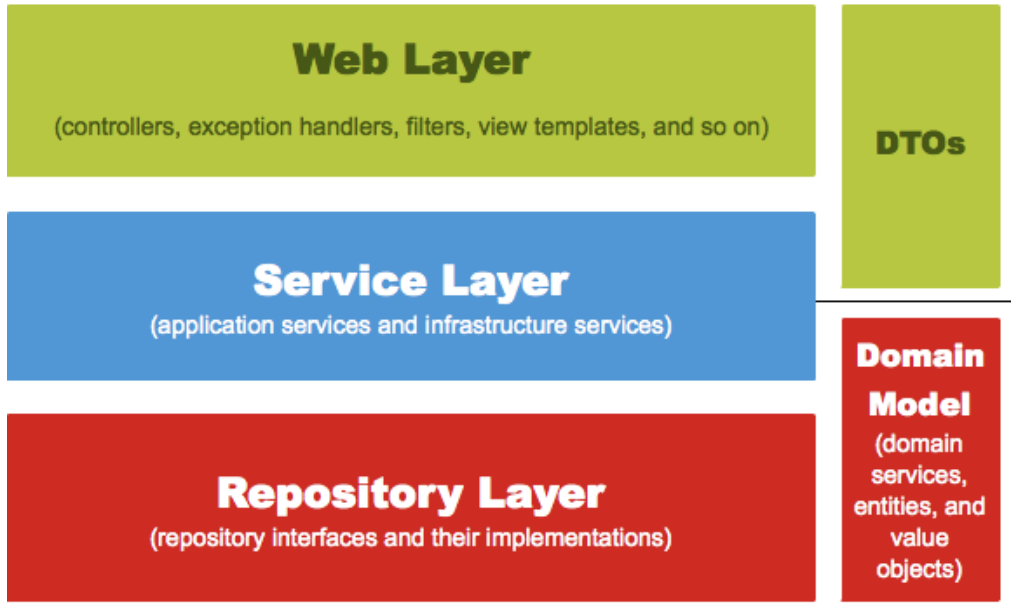


TDD across all Spring Layers

Integration Tests

@RunWith ->SpringRunner
@SpringBootTest

@Autowired
TestRestTemplate



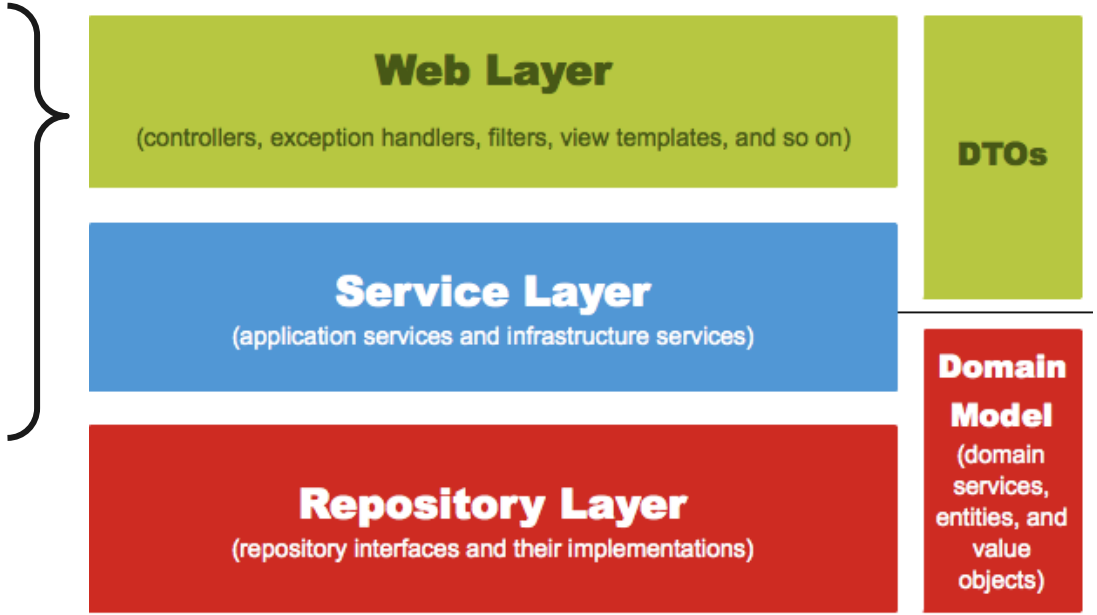
TDD across all Spring Layers

@RunWith ->SpringRunner.class
@WebMvcTest-> Controller.class

@Autowired
MockMvc

@MockBean
Service

Controller Tests



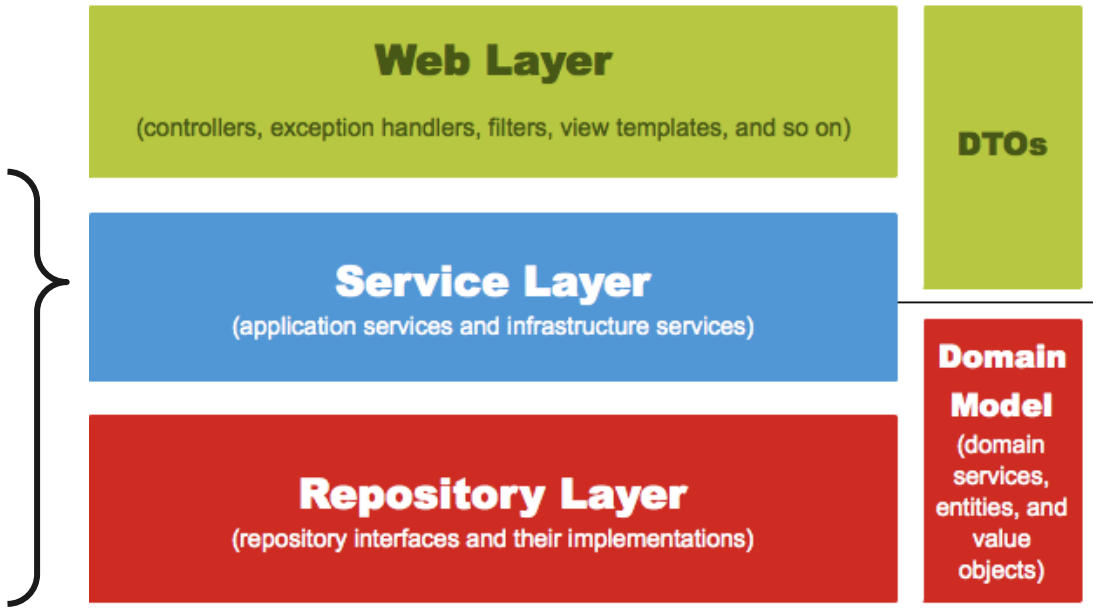
TDD across all Spring Layers

Service Tests

@RunWith ->MockitoJUnitRunner.class

@Mock
Repository

@InjectMocks
Service



TDD across all Spring Layers

Repository Tests

```
@RunWith ->SpringRunner.class
@DataJpaTest
@AutoConfigureTestDatabase(
        replace = NONE
)

@Autowire
Repository
```