

Test-driven Development

Pablo Rodriguez Massuh

Algoritmos y Programación III – Fiuba



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

Desarrollo guiado por pruebas

¿En qué consiste?

Práctica que involucra:

- **Escribir las pruebas primero.**
- **Refactorizar.**

¿Con qué objetivos?

- **Nos obliga a pensar a través de la interfaz necesaria, antes de desarrollar la funcionalidad.**

Escribimos la API como nos gustaría utilizarla a nosotros mismos.

Testeo Manual

Proceso de desarrollo con Testeo Manual

- 1) Escribir el código y compilarlo.
- 2) Subirlo al servidor (puede ser local) e iniciar la aplicación.
- 3) Correr el código manualmente. (En muchos casos será completar campos paso por paso manualmente).
- 4) Revisar archivos de LOG, Base de datos, valores de variables, impresiones por pantalla, etc.
- 5) Si no funcionó, repetir todo el proceso anterior.

Desventajas del Testeo Manual



¡¡ Es ABURRIDO !!

- 1) Nadie tiene ganas de llenar siempre el mismo formulario.
- 2) No tenemos nada nuevo para aprender testeando manualmente.
- 3) La gente tiene a *descuidarse* cuando los hace.
- 4) Nadie Mantiene una lista de los test necesarios a ser corridos manualmente.

Más Desventajas del Testeo Manual

- 5) Las pruebas manuales no son reutilizables.
- 6) Siempre consumen el mismo esfuerzo.
- 7) Sólo los *desarrolladores* pueden ver el resultado.
- 8) Los resultados de los tests deben ser vistos por distintos perfiles: *Desarrolladores, Gerentes, Team Leaders*.
- 9) El testeo manual se ejecuta post-facto convirtiéndolos solamente en *drive bug-patching*.
- 10) Ensucian el código de la aplicación y la vuelven más lenta al incorporar impresiones por consola, escritura de logs, etc.

Acerca de los frameworks xUnit

Creado por Kent Beck y Erich Gamma

- Diseñado originalmente para JAVA (JUnit), luego se extendió a otros lenguajes (xUnit). En el caso de SmallTalk SUnit.
- Cada test unitario debe probar una “unidad”.

Propone como buenas prácticas:

- Escribir muchas pruebas, al menos una para cada funcionalidad.
- Si un test falla, debería indicarnos dónde está el error.

Beneficios de las pruebas automatizadas

Una vez escritas, las pruebas deben poder correrse sin esfuerzo manual en reiteradas ocasiones.

Las pruebas nos indican cómo debe funcionar el sistema:

Cuando trabajamos en el código, las pruebas nos ayudan a verificar que no hemos roto nada, ya sea que hayamos:

- + Agregado nueva funcionalidad.**
- + Hecho algún refactor.**

Beneficios de las pruebas automatizadas

Documentación tiende a “juntar polvo”

-No nos daremos cuenta cuando esté desactualizada

Las pruebas nos indican cómo debe funcionar el sistema:

-Si las mismas quedan desactualizadas, se notará bastante rápido (La próxima vez que se corran)

Pasos Para TDD

- Escribir el test, el cual no compilará.
- Agregar el mínimo código posible para que compile.
- Correr el test, el cual fallará (Barra Roja).
- Luego agregar el mínimo código posible para que el test pase.
- Correr el test y verificar que el mismo pase (Barra Verde).

Entonces... ¿Ya estamos?



**FACULTAD
DE INGENIERIA**

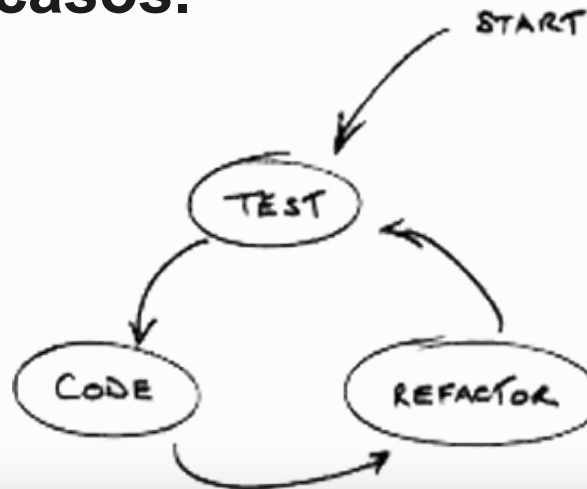
Universidad de Buenos Aires

fppt.com

Pasos Para TDD

¡ Noooo !

- Una vez que el test pase, debemos refactorizar el código para que quede más elegante.
- También deberíamos considerar ampliar nuestros test. Como las nuevas pruebas no pasarán, de seguro tendremos que modificar nuestro código para que abarque los nuevos casos.
- Repetir el proceso



Un ejemplo: crear una Puerta

- ***La puerta debe tener una manija:***

testPuertaDeberiaTenerManija

"La puerta debería tener una manija con la cual operar."

| puerta |

puerta := Puerta new.

self assert: puerta tieneManija.

- ***Si giro la manija y empujo la puerta, esta debería abrirse:***

testDeberiaAbrirseLaPuertaAlGirarLaManijaYEmpujar

| unaPuerta |

unaPuerta := Puerta new.

unaPuerta girarManija.

unaPuerta empujar.

self assert: unaPuerta estaAbierta.



FACULTAD
DE INGENIERIA

Universidad de Buenos Aires

Refinando el ejemplo

- ***Cuando la puerta esté abierta debo poder tirar de la manija y cerrarla:***

testPuedoCerrarLaPuertaTirandoDeLaManija

```
| unaPuerta |  
unaPuerta := Puerta new.  
unaPuerta girarManija.  
unaPuerta empujar.  
unaPuerta tirarManija.
```

self assert: (unaPuerta estaAbierta) not.

- ***Si empujo la puerta sin girar la manija debería permanecer cerrada:***

testEmpujoLaPuertaSinGirarManijaPuertaPermaneceCerrada

```
| puerta |  
puerta := Puerta new.  
puerta empujar.
```

self assert: (puerta estaAbierta) not.



Refinando más el ejemplo

testDeberiaNoAbrirLaPuertaSinGirarManijaDespuesDeAbrirlaUnaVez

| puerta |

"Abrimos la primera vez"

puerta := Puerta new.

puerta girarManija.

puerta empujar.

"Cerramos"

puerta tirarManija.

"Abriendo por segunda vez..."

puerta empujar.

self assert: (puerta estaAbierta) not.

¿ Preguntas ?



FACULTAD
DE INGENIERIA

Universidad de Buenos Aires

fppt.com