



TDD: Test-Driven Development

Leticia Cruz

TDD

**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

¿Por qué TDD?

- Es la manera más sencilla de una buena calidad en el código.
- Eliminas el miedo al cambio en el código, ya sea agregar nueva implementación o modificar la existente.
- La refactorización del código se vuelve más sencilla y más segura.
- Cuando hacemos un cambio en el código no necesitamos ir manualmente probando funcionalidad por funcionalidad para ver si no rompimos algo.
- El set de pruebas se puede automatizar cuando usamos pipelines de CI/CD.
- Nos permite escribir un código limpio (ser profesionales y desarrollar usando buenas prácticas)





¿Qué es TDD?

Es una técnica de desarrollo de software en donde primero se escribe una prueba que falla antes de que se escriba un código funcional.

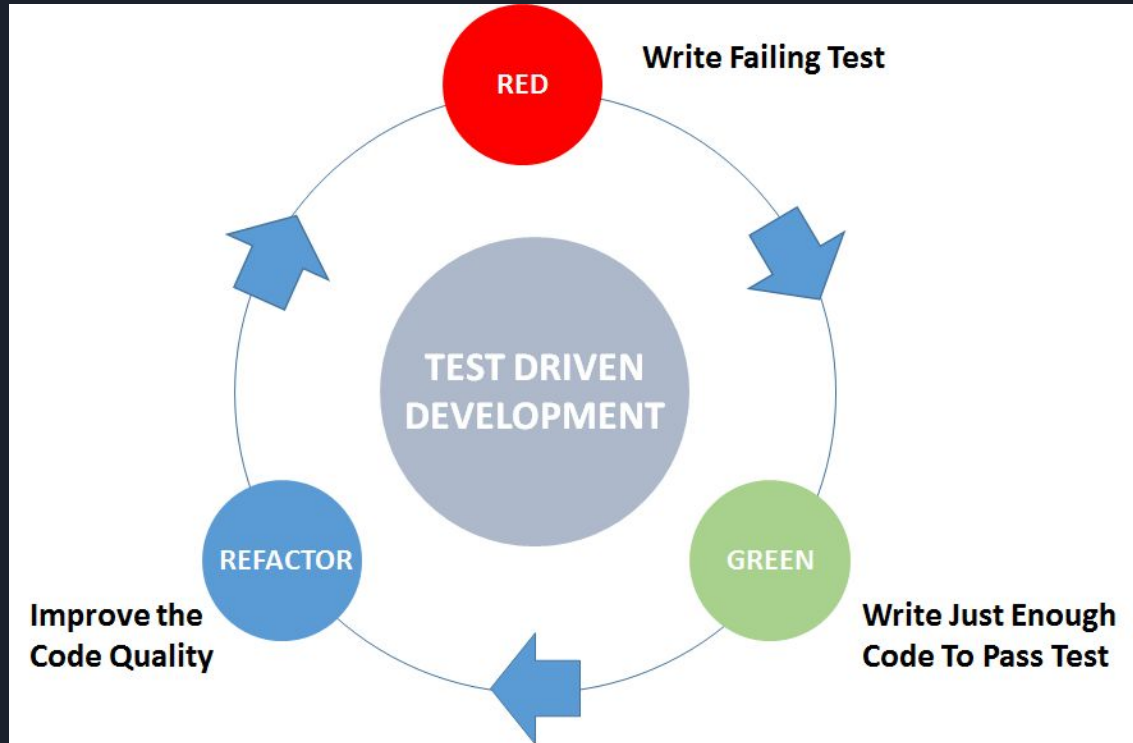
Esta técnica suele relacionarse con metodologías ágiles de desarrollo.



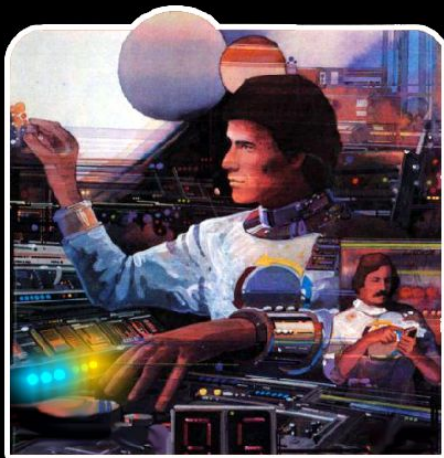
Las 3 leyes del TDD

- 01 Escribir el suficiente código de tal manera que tengas una prueba unitaria que esté fallando.
- 02 Escribir el suficiente código para lograr que pasen todas tus pruebas.
- 03 Refactorizar tu código hasta que esté sencillo y limpio.

Ciclo de TDD



THE TWO STATES OF EVERY PROGRAMMER



I AM A GOD.




**I HAVE NO IDEA
WHAT I'M DOING.**



Recomendaciones y Buenas Prácticas en TDD





Darle mantenimiento a nuestras pruebas y mantenerlas limpias

- Tener pruebas obsoletas es equivalente, sino es que peor, a no tener pruebas.
- Las pruebas deben de cambiar en la medida en que cambia el código en producción.
- Mientras haya más pruebas obsoletas, será más complicado cambiarlas.
- Las pruebas son tan importantes como nuestro código en producción: requiere de pensar, diseñar y poner atención en cómo las hacemos.



Buenas prácticas en TDD

- Tener un *assert* por prueba.
- Un concepto por prueba.
- F.I.R.S.T. (Reglas para tener unas pruebas limpias):
 - **Fast:** Las pruebas deberán de ejecutarse rápido.
 - **Independent:** Las pruebas deberán de poder ejecutarse independientemente unas de otras.
 - **Repeatable:** Las pruebas deberán de poder ejecutarse en cualquier ambiente: producción, testing, local.
 - **Self-validating:** Las pruebas deberán de tener una salida verdadera o falsa, es decir, pasó o no pasó la prueba.
 - **Timely:** De ser posible las pruebas deben ser desarrolladas antes de empezar a escribir el código de la aplicación.

Consideraciones Finales

Murphy's Law of Debugging: The thing you believe so deeply can't possibly be wrong so you never bother testing it is definitely where you'll find the bug after you pound your head on your desk and change it only because you've tried everything else you can possibly think of.

- TDD requiere mucho más tiempo que la programación "normal".
- ¿Cuántas pruebas debería de escribir?
- ¿Mi cobertura de las pruebas debería ser del 100%? No siempre cantidad es calidad.
- Soy capaz de escribir código con muy pocos errores, no necesito TDD.





TDD OR CODE FIRST?



memegenerator.net