🧬 TDD, desarrollo guiado por las pruebas

Todo empieza definiendo una prueba.

> *"Haz lo más simple que pueda funcionar."*
>
> -- ✍️ Kent Beck

- Si haces las pruebas antes... bien, porque al menos tienes **las pruebas hechas**.
- Solo tienes que hacer **el mínimo código** que pase la prueba. Nada más.
- Para poder probar fácilmente, harás un código fácil de manejar; **mejor diseñado**.

Todo empieza con los requerimientos

Saber qué vamos a hacer

historias de programador.

Son pruebas de **caja blanca** y su documentación es técnica y precisa.

```
FEATURE: a BankClient account
As_a: high level service
I_want_to: have a class where deposit money
In_order_to: accumulate several amounts of money for later
```

Aún muy genérico, mejorarlo especificando casos de comportamiento esperado.

```
Given: a new BankClient objec
When: i make a deposit of 10
Then: returns a balance of 10
```

Cadenas de texto que acompañan a las pruebas

Las pruebas

Las pruebas **TDD son pruebas para programadores**. Las hacemos por nuestro propio bien. Sin que nos las pidan, sin esperar que las valoren.

> Hacemos las pruebas para estar seguros de hacer lo que se pide, nada más, pero bien hecho.

La estructura, los textos y el cómo se hacen debe ser a nuestro gusto. Yo te propongo seguir con la **estructura *AAA* y el nombrado *GWT***.

```
describe('GIVEN: a new BankClient object', () => {
  const sut = new BankClient();
  test('WHEN: i make a deposit of 10 THEN returns a balance of 10', () => {
    const input = 10;
    const actual = sut.deposit(input);
    const expected = 10;
    expect(actual).toEqual(expected);
  });
});
```

Y la ejecutamos... y falla. 🔴

La implementación

Ahora que hemos visto fallar a nuestra prueba, vamos a hacer que la pase. ¿Cómo? Escribiendo **el mínimo código que satisfaga** la especificación funcional descrita.

```
export class BankClient {
  constructor() {}
  deposit(amount) {
    return 10;
  }
}
```

Listo 🟩, vámonos a casa que se está haciendo de noche.

La mejora

Es momento de hacer dos cosas. Lo primero enriquecer las pruebas, lo segundo refactorizar el código.

```
import { BankClient } from './bank-client';
describe('GIVEN: a new BankClient object', () => {
  let sut;
  beforeEach(() => {
    sut = new BankClient();
  });
  test('WHEN: i make a deposit of 10 THEN returns a balance of 10', () => {
    const input = 10;
    const actual = sut.deposit(input);
    const expected = 10;
    expect(actual).toEqual(expected);
  });
  test('WHEN: i make a deposit of 15 THEN returns a balance of 15', () => {
    const input = 15;
    const actual = sut.deposit(input);
    const expected = 15;
    expect(actual).toEqual(expected);
  });
});
```

Ok, ya veo dónde falla 🔴.

```
export class BankClient {
  constructor() {}
  deposit(amount) {
    return amount;
  }
}
```

Ahora sí que está bien 🟩.

No tan rápido, vamos a seguir enriqueciendo la prueba. Agrega este caso

```
test('WHEN: i make a deposit of 10 and another of 15 THEN the last one returns 25', () => {
  let input = 10;
  sut.deposit(input);
  input = 15;
  const actual = sut.deposit(input);
  const expected = 25;
  expect(actual).toEqual(expected);
});
```

Vaya 🔴, parece que necesitaré algún tipo de acumulador...

```
export class BankClient {
  constructor() {
    this.acumlator = 0;
  }
  deposit(amount) {
    this.acumlator += amount;
    return this.acumlator;
  }
}
```

Correcto de nuevo 🟩

El ciclo virtuoso

Este ciclo descrito se completa con un proceso de ***refactoring*, o mejora en el diseño**. Este trabajo se realiza sobre el código correcto; lo recalco, **es una mejora**.

```
export class BankClient {
  constructor() {
    this.balance = 0;
  }
  deposit(amount) {
    this.balance += amount;
    return this.balance;
  }
}
```

Pequeñas mejoras constantes

🔴 RED : definir la prueba y comprobar que falla.

🟩 GREEN : Escribir el mínimo código posible que satisfaga la prueba.

💙 REFACTOR : Mejorar dicho código manteniendo el respaldo de la prueba.

> Repetir este ciclo refinando y creando nuevas pruebas hasta completar el requerimiento funcional completo.