TDD - Test Driven Development

Desarrollo guiado por pruebas

Qué es TDD?

Qué es TDD?

TDD es una práctica de programación que propone **invertir el orden tradicional** en que desarrollamos software:

En lugar de escribir un programa y luego escribir pruebas para “verificar” que funciona bien, propone en cambio ***primero escribir las pruebas, y después escribir el código fuente que haga pasar las mismas***.

Qué es TDD?

La práctica de TDD es inventada (redescubierta?) por Kent Beck, y plasmada en varios libros a fines de los 90's.

Esta técnica forma parte, a su vez, de la metodología de desarrollo de software conocida como ***eXtreme Programming*** o simplemente *XP*.

Otra práctica usualmente utilizada en conjunto con TDD es la ***programación en parejas***.

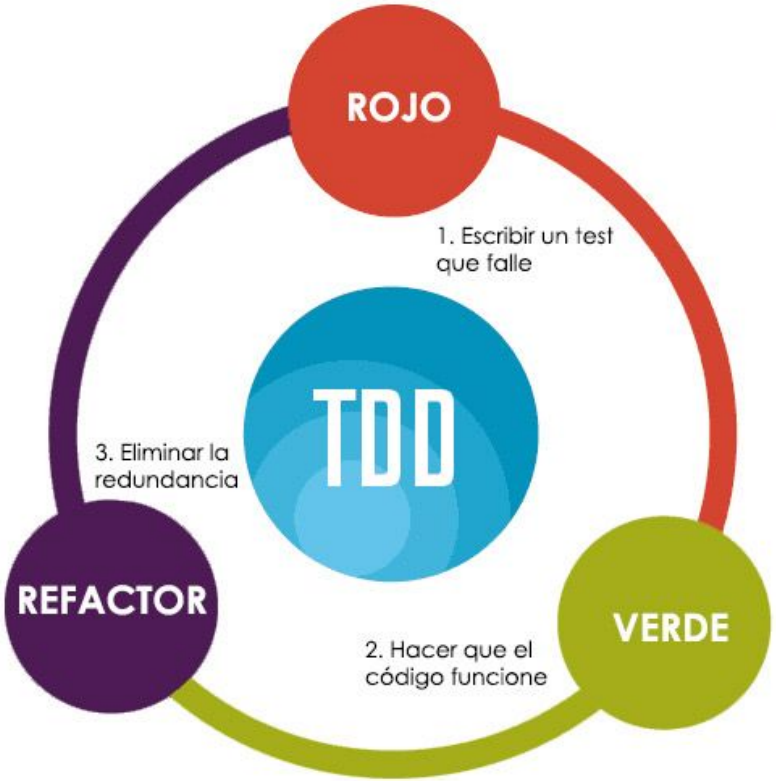
En qué consiste hacer TDD?

En qué consiste hacer TDD?

La práctica de TDD es muy simple. Comienza llevando a cabo los siguientes pasos:

1. *Escribir una prueba que falle.*
2. *Escribir la mínima cantidad de código que haga pasar mi prueba.*
3. *Reflexionar sobre lo escrito, y mejorar la organización del código si es necesario.*

Estos tres pasos conforman una iteración del ciclo de TDD. Estas iteraciones se llevan a cabo una y otra vez hasta que el desarrollo alcanzado sea satisfactorio.



Por los colores utilizados normalmente en los frameworks de testing, al ciclo de TDD también se lo conoce como:

RED - GREEN - REFACTOR

Fase “ROJO”

- Al momento de elegir una prueba, se debe elegir aquella que sea más fácil de escribir, más fácil de entender, y que apunte a conocer algún aspecto relevante sobre la solución que estoy buscando.
- La prueba puede fallar por diferentes motivos:
  - Errores durante la compilación del programa
  - Errores durante la ejecución del programa
  - Errores de lógica en el programa

Fase “VERDE”

- Al momento de escribir código para hacer pasar mi test, elegir la solución más simple y directa que se me ocurra.
- En la medida en que sea posible, intentar no borrar código ya existente, y sólo agregar código nuevo.
- *En esta fase es posible “hacer trampa” y escribir valores arbitrarios que hagan pasar la prueba.*

Fase “REFACTOR”

En esta fase se intenta **mejorar el código** escrito (tanto de pruebas como de la solución) **sin agregar, quitar, ni modificar su funcionalidad**. Por ejemplo:

- Eliminar código repetido
- Añadir modularización
- Renombrar variables y/o funciones
- Introducir alguna generalización/abstracciones (algo2!)

Beneficios de usar TDD

Beneficios de usar TDD

Por la naturaleza de sus etapas, TDD nos invita a invertir tiempo en cosas que no siempre lo hacemos, y que son ampliamente beneficiosas durante el proceso de desarrollo de software.

TDD nos invita a...

- pensar en qué queremos que cumpla nuestro software *antes* de empezar a desarrollarlo.
- dividir en partes más pequeñas aquellas funcionalidades que nos resulten más complejas de probar.
- generar documentación que dé cuenta de cómo funciona y cómo se usa el software construido.
- construir un mecanismo que permita y facilite la detección de errores en nuestro sistema.

Ejemplo

Ta-Te-Ti

Objetivo: escribir una función que dado un tablero de tateti válido devuelve el estado del juego.

El tablero se representa mediante una matriz de enteros. Cada posición contiene el número del jugador que ocupa esa casilla (1 ó 2), ó 0 (cero) si la posición está libre.

El estado del juego se representa con un entero, que puede ser: 1 o 2, indicando que ese jugador ha ganado; 0 (cero) si el juego terminó en empate; o -1, si el juego aún no ha finalizado.

Ta-Te-Ti

Ejemplos:

- `tateti({{1,2,2},{1,1,2},{1,0,0}) ->  1`
- `tateti({{2,2,2},{1,1,0},{1,0,0}) ->  2`
- `tateti({{2,1,2},{2,1,2},{1,2,1}) ->  0`
- `tateti({{0,0,2},{0,2,1},{0,0,1}) -> -1`

Resolución de Ta-Te-Ti en clase

Ejercicios

FizzBuzz

Objetivo: escribir una función que dado un número natural devuelve “fizz” si es múltiplo de 3, “buzz” si es múltiplo de 5, “fizzbuzz” si es múltiplo de ambos, o el mismo número en caso contrario.

Ejemplos:

- `1 -> “1”`
- `3 -> “fizz”`
- `5 -> “buzz”`
- `15 -> “fizzbuzz”`

Factores Primos

Objetivo: escribir una función que dado un número natural, devuelva una lista con la descomposición en sus factores primos.

Ejemplos:

- `factoresPrimosDe( 1 ) -> [ ]`
- `factoresPrimosDe( 1 ) -> [ 2 ]`
- `factoresPrimosDe( 9 ) -> [ 3, 3 ]`
- `factoresPrimosDe( 56 ) -> [ 2, 2, 2, 7 ]`

Bibliografía

Bibliografía adicional

- “Extreme Programming Explained”
  - Kent Beck - 1999
- “Refactoring: Improving the design of existing code”
  - Kent Beck, Martin Fowler - 1999
- Test-Driven Development by Example
  - Kent Beck - 2000

Preguntas?

Muchas gracias!