Departamento de
Ciencias de la Computación y Tecnologías de Información
Universidad del Bío-Bío

Desarrollo conducido por pruebas, TDD

Tópicos Avanzados en Ingeniería de Software, MCC

María Antonieta Soto Ch.
2019-1

1

TDD

- El Desarrollo Conducido (o Dirigido) por Pruebas (Test Driven Development TDD), es una técnica de diseño e implementación de software propuesta por Kent Beck.
- TDD es una técnica para diseñar software que se centra en tres pilares fundamentales:
  - La implementación solo de las funciones que el cliente necesita y nada más.
  - La minimización del número de defectos que llegan al software en fase de producción.
  - La producción de software modular, altamente reutilizable y preparado para el cambio.

2

2

TDD

- No se trata de escribir pruebas a granel, sino de diseñar adecuadamente según los requisitos.
- Se pasa de pensar en implementar tareas, a pensar en ejemplos certeros que eliminen la ambigüedad creada por la prosa en lenguaje natural.
- Hasta ahora las tareas, o los casos de uso, eran las unidades de trabajo más pequeñas sobre las que ponerse a desarrollar código.

3

TDD

- Con TDD se intenta traducir el caso de uso o tarea en X ejemplos, hasta que estos sean suficientes para describir la tarea sin lugar a malinterpretaciones de ningún tipo.
- En TDD la propia implementación de pequeños ejemplos, en constantes iteraciones, hace emerger la arquitectura que se necesita usar.

4

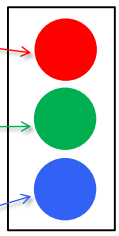
TDD. Un ejemplo

- Una funcionalidad que abona un monto de dinero a una cuenta bancaria:
  - "al abonar 10.000 pesos a una cuenta con saldo 50.000, el nuevo saldo deberá ser 60.000"
  - "al abonar 0 pesos a una cuenta con saldo 50.000, el nuevo saldo deberá ser 50.000"
  - "al abonar -10.000 pesos a una cuenta con saldo 50.000, el nuevo saldo deberá ser 40.000"
  - …

5

TDD

- Proceso de 3 pasos:
  - Paso 1: Hacer que falle
    - Ningún código debe ser escrito sin un test que falle.
  - Paso 2: Hacer que funcione
    - Escribir el código tan simple como sea posible para que pase el test.
  - Paso 3: Hacerlo Mejor
    - Refactorizar el código y probar el test de modo que esto sea lo mejor que se pueda hacer.



6

TDD

- Escribir el ejemplo o test primero
  - Un test inicialmente es un ejemplo o especificación de una funcionalidad. Esta especificación no es estática, un test se puede modificar.
  - Para poder escribirlo, tenemos que pensar primero en cómo debe ser el comportamiento de la funcionalidad.

7

TDD

- Escribir el ejemplo o test primero
  - Pero sólo una parte pequeña, un comportamiento bien definido y sólo uno. Tenemos que hacer el esfuerzo de imaginar cómo seria el código de la funcionalidad si ya estuviera implementado y cómo comprobaríamos que, efectivamente, hace lo que le pedimos que haga.
  - El hecho de tener que usar una funcionalidad antes de haberla escrito le da un giro de 180 grados al código resultante.

8

TDD

- Implementar el código que hace funcionar el ejemplo

  Teniendo el ejemplo escrito, se codifica lo mínimo necesario para que este se cumpla, para que el test pase (el de menor número de caracteres).
  - No importa que el código se vea "feo", eso se va a enmendar en el siguiente paso y en las siguientes iteraciones.

9

TDD

- Implementar el código que hace funcionar el ejemplo
  - La máxima es no implementar nada más que lo estrictamente obligatorio para cumplir el ejemplo actual. Y no se trata de hacerlo sin pensar, sino concentrados para ser eficientes.
  - Parece fácil, pero al principio no lo es, siempre se escribe más código del que hace falta…

10

TDD

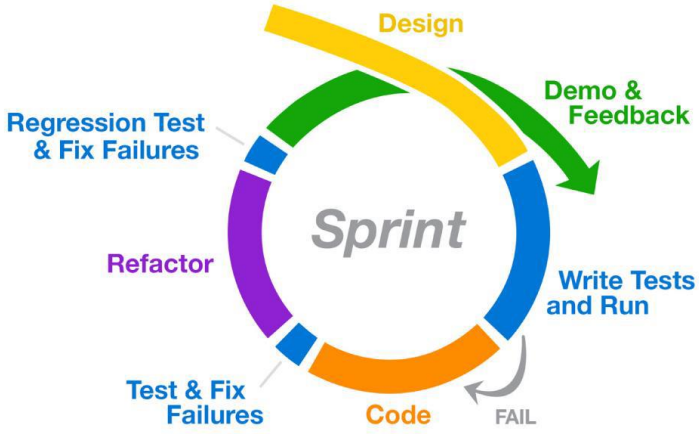
- Refactorizar
  - No significa reescribir el código; reescribir es más general que refactorizar.
  - Es modificar el diseño sin alterar su comportamiento. De ser posible, sin alterar su API pública.
  - Se revisa el código (también el del test) en busca de líneas duplicadas y se eliminan refactorizando.
  - También revisar el código para hacerlo más claro y fácil de mantener.

Pasos genéricos en TDD

- **Escribir un Test, aún de algo que todavía no hayamos codificado**,
- **Compilar**, va a dar error, ya que todavía no hemos codificado lo que estamos probando,
- **Escribir la mínima cantidad de código para que compile**, aunque sea la cabecera del método o función a probar,
- **Compilar**, ahora no da error, estamos en condiciones de ejecutar nuestro Test,
- **Ejecutar el Test**, falla, ya que habíamos escrito la mínima cantidad de código que garantice la compilación, pero no la ejecución del Test,
- **Escribir la mínima cantidad de código para que el Test no falle**,
- **Ejecutar el Test**, esta vez no falla,
- **Refactorizar**.

- **Recomenzar escribiendo un nuevo Test**.

TDD y el desarrollo ágil



13

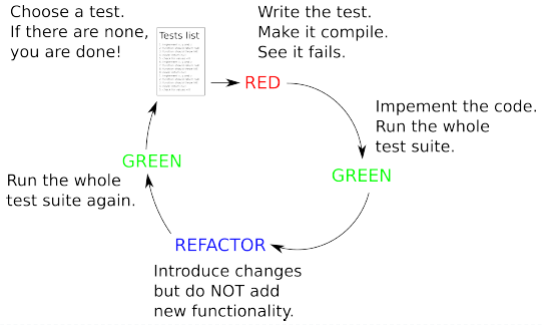
TDD y la automatización

- Uno de los pilares de TDD es la automatización.
- En el mercado hay diferentes herramientas que lo soportan, algunas de ellas:

14

TDD, consideraciones adicionales

- Tome en cuenta la siguiente regla:
  - ¡Nunca escriba código no asociado a un test que haya fallado!
  - Termine el ciclo estando en verde.
  - El ciclo termina (es decir, alguna funcionalidad se ha implementado) cuando no hay más pruebas que escribir.



15

15

¿Cómo elegir el siguiente test a escribir?

- **Lo más fácil primero**
  - Esta regla dice: "Empezar con algo realmente simple. Implementar un test obvio."
  - Ejemplos de este tipo son:
    - Escribir una prueba de comprobación de parámetros para una función (no importa cuál sea el propósito de la función en cuestión).
    - Escribir un parser, comenzar con el caso de prueba consistente en pasar una cadena vacía al método de análisis y recibir null como retorno.

16

16

¿Cómo elegir el siguiente test a escribir?

- **Lo más fácil primero**
  - En ninguno de los dos casos se tocará la lógica principal (probablemente bastante compleja) que debe probarse e implementarse.
  - Sin embargo, se terminaría con algunas clases y métodos que podrían dar una idea de la verdadera tarea por delante. Esto podría ser muy útil, al sentirse perdido y no saber cómo continuar.

17

¿Cómo elegir el siguiente test a escribir?

- **El más informativo**
  - Otra aproximación es comenzar con el que da la mayor información sobre la funcionalidad a implementar.
  - La idea es obtener lo máximo posible en términos de conocimiento.

18

¿Cómo elegir el siguiente test a escribir?

- **El caso típico primero, luego los demás**
  - Parece bastante razonable comenzar con un "caso típico".
  - Se debe pensar en el uso más probable de la funcionalidad.
  - Ejemplos:
    - Al escribir un tokenizer, comenzar con una oración válida como una entrada.
    - Al implementar una máquina expendedora, comenzar con un cliente que inserta una moneda de $500 y selecciona un producto que la máquina tenga disponible.
  - Más adelante, se implementan los casos de borde.

19

¿Cómo elegir el siguiente test a escribir?

- **Escuchar la voz de su propia experiencia**
  - Probablemente la manera más valiosa de lidiar con el dilema del "próximo test" es escuchar la propia experiencia.

20

Bibliografía

- Carlos Blé Jurado y colaboradores. Desarrollo Ágil con TDD. Primera Edición. Licencia Creative Commons (www.iExpertos.com), 2010. Cap. 2 y 4.
- Tomek Kaczanowski. Practical Unit Testing with JUnit y Mockito, Kackzanowscy.pl Tomasz Kaczanowski, 2013. Cap. 3 y 4.