UNIVERSIDAD PRIVADA DE TACNA

INGENIERIA DE SISTEMAS

TEMA:

Test-Driven Development

CURSO:

BASE DE DATOS II

DOCENTE(ING):

Patrick Jose Cuadros Quiroga

Integrantes:

Marko Antonio RIVAS RIOS	(2016055461)
Jorge Luis MAMANI MAQUERA	(2016055236)
Andree Ludwed VELASCO SUCAPUCA	(2016055286)
Yofer Nain CATARI CABRERA	(2017059289)
Adnner Sleyder ESPERILLA RUIZ	(2015050543)
Jesus ESCALANTE ALANOCA	(2015050641)

Índice

1. Objetivos del Desarrollo Orientado A Pruebas (TDD)

El propósito de esta técnica tiene los siguientes 3 objetivos básicos:

1. Minimizar el número de bugs: Mientras más bugs salgan menos rentable es el proyecto, porque corregir los bugs es tiempo que se puede invertir mejor en otras tareas. Si no hay bugs se puede conseguir una mayor rentabilidad de la aplicación.

2. Implementar las funcionalidades justas que el cliente necesita:

Es muy común que cuando se explican los requisitos de una aplicación o las especificaciones en la fase de análisis y diseño se esboza una aplicación, y el diseñador por su experiencia con funcionalidades pensando que van a ser útiles para la aplicación, para el cliente o para otros componentes. Sin embargo casi el 95 % de esas funcionalidades extras no se usan en ninguna parte de la aplicación, eso implica tiempo invertido desarrollando algo que no ha llegado a nada. El objetivo de TDD es eliminar ese código innecesario y esas funcionalidades que no ha pedido el cliente, con lo cual redunda en eficiencia, tanto en el trabajo y como en la rentabilidad de la aplicación.

3. Producir software modular, altamente reutilizable y preparado para el cambio: Esto es realmente más técnica, porque con buenos hábitos de programación siempre se logra que el proyecto sea modular y reutilizable. Prepararlo para el cambio es una característica que no se consigue siempre y que con TDD si, ya que muchas veces cuando se tiene que cambiar la funcionalidad de la aplicación se tiene que refactorizar código ajeno, trabajar con código complicado, entre otras cosas; en cambio con TDD se tiene la confianza de que cuando se haga cambios no se van a estropear las funcionalidades que ya se tienen. Esto se consigue ya que la forma funcional de TDD es que primero se construye la prueba y luego el código hace que todo lo que surja en él ya este testeado, así que cualquier cambio que se vaya a introducir estará cubierto por los tests y si llegas a dañar algo alguno de ellos reaccionara cuando se ejecuten.

Todo esto cambia un poco la mentalidad tradicional que es: primero analizar los requisitos, luego hacer un diseño completo y profesional, después empezar a codificar y por último testear. Lo que hay que hacer es que, en vez de planear tareas pensar en ejemplos y datos concretos, ya que en eso se basa los test, tener parámetros de entrada y de salida y luego ver si la respuesta es lo que se esperaba. Si con TDD consigues tener en las especificaciones o requisitos una lista de ejemplos muy completa, que sean concretos, que elimine cualquier tipo de ambigüedad y que se puedan transformar en pruebas, al final tendrás una batería de tests que te cubrirán todas las funcionalidades y el código resultante estará 100 % cubierto por dichos test.

2. Desarrollo Orientado A Pruebas (TDD)

1. ¿Qué es Desarrollo Orientado A Pruebas (TDD)?

Esta técnica llamada TDD (Test Driven Development), se puede definir como un proceso de desarrollo de software que se basa en la idea de desarrollar unas pequeñas pruebas, codificarlas y luego refactorizar el código que hemos implementado anteriormente. Podemos decir que esta técnica e implementación de software está dentro de la metodología XP donde deberíamos de echarle un ojo a todas sus técnicas, tras leer varios artículos en un coincido con Peter Provost con un diseño dirigido o implementado a base de ejemplos hubiese sido mejor pero TDD se centra en 3 objetivos claros:

- Una implementación de las funciones justas que el cliente necesita y no más, solamente las funciones que necesitamos, estoy cansado de duplicar dichas funciones para que hagan lo mismo

  Mínimos defectos en fase de producción

- Producción de software modular y sobre todo reutilizable y preparado para el cambio

  Esta técnica se basa en la idea de realizar unas pruebas unitarias para un código que nosotros debemos construir, Nuestro TDD lo que nos dice es que primero los programadores debemos realizar una prueba y a continuación empezar a desarrollar el código que la resuelve. El método que debemos seguir a para empezar a utilizar TDD es sencillo, Nos sirve para elegir uno de los requisitos a implementar, buscar un primer ejemplo sencillo, crear una prueba, ejecutarla e implementar el código mínimo para superar dicha prueba. Obviamente la gracia de ejecutar la prueba después de crearla es ver que esta falla y que será necesario hacer algo en el código para que esta pase. El ciclo de desarrollo de TDD es empezar la prueba, en test realizar un test, revisar el código y pasar el refactor.

  Crear la prueba o test

- Ejecutar los tests: falla (ROJO)
- Crear código específico para resolver el test
- Ejecutar de nuevo los tests: pasa (VERDE)
- Refactorizar el código
- Ejecutar los tests: pasa (VERDE)

  Personalmente, añadiría lo siguiente:

- Incrementa la productividad.
- Nos hace descubrir y afrontar más casos de uso en tiempo de diseño.
- La jornada se hace mucho más amena.
- Uno se marcha a casa con la reconfortante sensación de que el trabajo está bien hecho.

  Ahora bien, como cualquier técnica, no es una varita mágica y no dará el mismo resultado a un experto arquitecto de software que a un programador junior que está empezando. Sin

embargo, es útil para ambos y para todo el rango de integrantes del equipo que hay entre uno y otro. Es una técnica a tener en cuenta en el desarrollo web y sobre todo en el desarrollo de ingeniería software donde debemos tener en cuenta muchos fallos antes de pasar a producción.

3. Ventajas del TDD

- Puedes mejorar el código de tu aplicación en cualquier momento sin miedo a que dañes algo, ya que las pruebas ya las tienes listas y deberán pasar siempre.
- Los test que realizamos sobre las interfaces de nuestra app no siempre son completos, generalmente es lo que nos acordamos probar.
- Los equipos de testing, development y analyst serán más felices.
- La lectura del código será mucho mejor al tener ejemplos de uso (las pruebas).

4. Test Driven Development (TDD)

1. ¿Qué es el TDD?
Es una metodología de desarrollo cuyo objetivo es crear primero las pruebas y luego escribir el software. Es decir, desarrollo guiado por pruebas.

2. TDD o Test-Driven Development

(desarrollo dirigido por test) es una práctica de programación que consiste en escribir primero las pruebas (generalmente unitarias), después escribir el código fuente que pase la prueba satisfactoriamente y, por último, refactorizar el código escrito. Con esta práctica se consigue entre otras cosas: un código más robusto, más seguro, más mantenible y una mayor rapidez en el desarrollo.

3. Se divide en 3 sub-prácticas

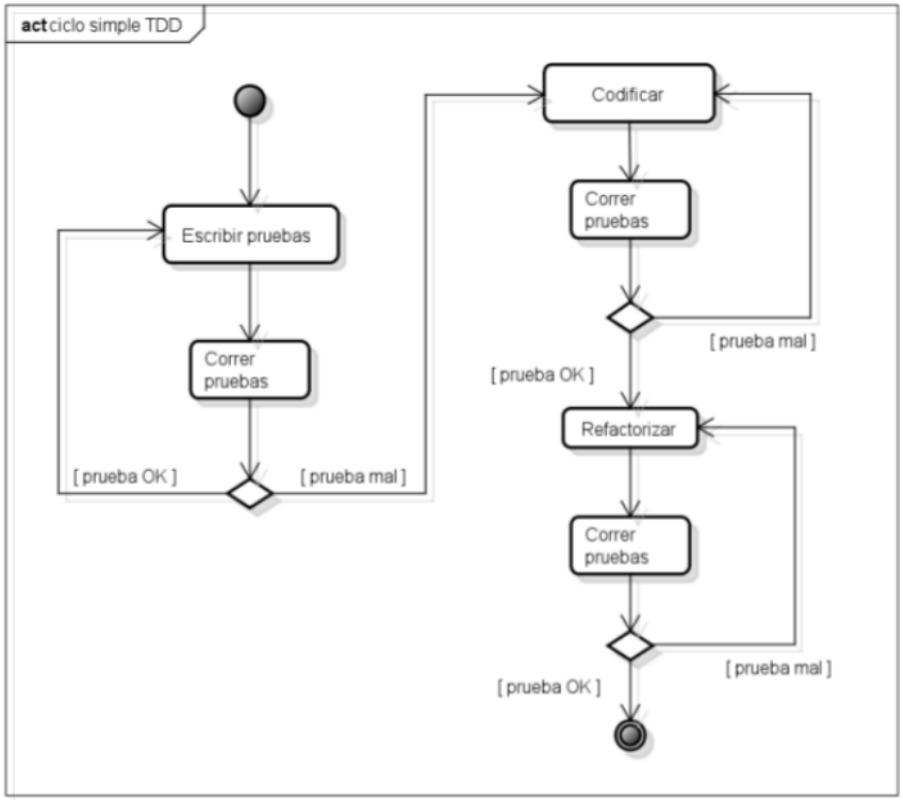
Test-First: las pruebas se escriben antes de escribir el propio código, y las mismas son escritas por los propios desarrolladores, esto busca que los mismos logren un entendimiento de lo que deben desarrollar mediante la construcción del código que lo va a probar.

Automatización: las pruebas deben ser escritas en código, y esto permite que se ejecuten automáticamente las veces que sea necesario, y el solo hecho de ejecutar las pruebas debe mostrar si la ejecución fue correcta o no.

Refactorizar el código: permite mantener la calidad de la arquitectura, se cambia el diseño sin cambiar la funcionalidad, manteniendo las pruebas como reaseguro.

Pruebas Unitarias y Simulación de objetos

Otra alternativa para realizar la simulación, consiste en conseguir probar el código unitariamente, esto significa aislarse de todos los recursos externos, es decir no depender de la infraestructura de red, o de un determinado entorno, o incluso del proceso que ejecutará nuestro código cuando esté en producción. Cargaremos las pruebas y el código a probar en un proceso encargado de gestionar la ejecución de las pruebas.



Tipos de Test

Test de aceptación: es un test que permite comprobar que se está cumpliendo con un requerimiento del negocio. Son pruebas escritas en lenguaje del cliente pero que puede ser ejecutado con la máquina. Esto nos permite probar que el software que estamos desarrollando cumple con las expectativas del cliente y de los usuarios.

Test funcionales: si bien, siendo estrictos, todos los tests son funcionales ya que prueban alguna funcionalidad, esta expresión es utilizada para determinar a aquellas pruebas que agrupan a varios tests de aceptación y prueban alguna funcionalidad del negocio propiamente dicha.

Test de sistema: integra varias partes del sistema, incluso puede probar toda la aplicación o varias funcionalidades juntas. Estos tests se comportan de manera similar y buscan emular el comportamiento de los usuarios del sistema.

Test unitarios: son los tests ineludibles, son los necesarios y los más importantes para los desarrolladores, todo test unitario debe ser rápido, atómico, inocuo e independiente, sino cumple con estas cuatro premisas no es un test unitario.

- La implementación de las funciones justas que el cliente necesita y no 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.

5. Profundizar tema

6. profundizar temaa

7. Webgrafía

- https://uniwebsidad.com/libros/tdd/capitulo-2
- https://docs.microsoft.com/es-es/previous-versions/bb932285(v=msdn.10)
- http://www.conaiisi.unsl.edu.ar/portugues/2013/158-524-1-DR.pdf