(Godoya, y otros, 2012)

Simulando Proyectos de Desarrollo de Software que incorporan Test Driven Development

En este artículo se presenta una línea de investigación denominada Simulación como Herramienta para la Mejora de los Procesos de Software Desarrollados con Metodologías Ágiles Utilizando Dinámica de Sistemas, cuyo objetivo es construir modelos de simulación utilizando la disciplina de dinámica de sistemas, que permitan estudiar los procesos de desarrollo de software agiles gestionados por diversas metodologías y prácticas ágiles.

Estos antecedentes brindan un Modelo de Simulación Dinámico para estudiar la incorporación de prácticas de Test Driven Development a los proyectos de software. Las encuestas realizadas en este artículo, brindan información importante sobre la acogida de la implementación de la técnica, TDD en una comunidad de desarrolladores de software.

5.2 Marco Conceptual

5.2.1 Test Driven Development (TDD).

Una práctica de desarrollo de software que se enfoca en Diseñar las pruebas de software antes de iniciar con el proceso de crear el código, el cual debe superar las pruebas diseñadas para él. (Beck, Test-Driven Development By Example, 2002)

5.2.2 Acceptance Test Driven Development (ATDD)

Nivel de TDD en el cual todo el equipo discute en colaboración criterios de aceptación, con ejemplos, y luego los divide en un conjunto de pruebas de aceptación en concreto antes de que comience el desarrollo. (Jurado, 2010)

5.2.3 Developer TDD

Nivel donde se diseñan las pruebas unitarias. Su objetivo es especificar a detalle, el diseño ejecutable para la solución. (Jurado, 2010)

5.3 Marco Teórico

La técnica TDD (Test Driven Development) se compone de la siguiente estructura.

Figura 1 Ciclo de vida TDD

5.3.1 Fases TDD

La práctica de TDD se divide en 3 fases, como se puede apreciar en la Figura 1 Ciclo de vida TDD. (Jurado, 2010)

- **Escribir las pruebas:** Esta etapa del ciclo también se le conoce como Rojo, hacer que la prueba falle. Las pruebas se escriben antes de empezar a escribir el código, la prueba debe buscar que haya una falla, estas son escritas por los propios desarrolladores una vez tengan claro el requisito y se busca que los mismos logren un entendimiento de lo que deben desarrollar mediante la construcción del código con el que se va a probar el test pase.

Se deben escribir en forma de Test unitarios [14] ya que se probara el código que satisfice el test, usualmente se utiliza un framework xUnit, pero xUnit no es una herramienta en sí misma. La letra x es tan solo un prefijo a modo de comodín, ya que son numerosos frameworks basados en el original SUnit. Sunit fue creado por Kent Beck para la plataforma SmallTalk y se ha llevado a otros lenguajes como Java, .Net, Python, Ruby, Perl, C++, etc.

- **Implementar el código que hace funcionar el ejemplo:** Esta etapa del ciclo también se le conoce como Verde, porque las pruebas no deben fallar al momento de implementar el código.

  Una vez que se escribe el test unitario en el framework elegido de acuerdo al lenguaje sobre el que se implementara la aplicación, se procede a escribir el código estrictamente necesario para que el test no genere error, y el cual hace énfasis en la expresión “Keep It Simple, Stupid!” (Déjelo Simple), se debe ignorar si el código se ve feo.

  Si bien se vendrán dudas al momento de implementar el código referente al comportamiento concerniente a ciertas variables de entrada y salida, esto se debe escribir para tenerlo en cuenta en una próxima iteración, más no se debe desarrollar en la iteración actual.

- **Refactorización del código:** Permite mantener la calidad de la arquitectura, se cambia el diseño sin cambiar la funcionalidad, y no debe afectar la ejecución de los tests unitarios. Se buscan las líneas duplicadas y se eliminan, aplicando la refactorización.

5.3.2 ATDD (Acceptance Test Driven Development)

Con respecto a ATDD, se pueden definir las siguientes características (Jurado, 2010):

- Historias de Usuario. Se escribe el requisito especificando el rol, lo que requiero y para que lo requiero, luego se escriben los ejemplos que vendrían siendo los criterios de aceptación, esto se lleva a cabo por medio del artefacto Historias de Usuario y Criterios de aceptación.

- Discutir. Un probador, desarrollador y experto en dominios se sientan con el propietario del producto y discuten la historia, descomponiéndola y generando las especificaciones o criterios de aceptación.

- Decidir. El propietario del producto decide que las especificaciones cubren el comportamiento.

- Desarrollo. El equipo de tester anota las especificaciones olvidas para la siguiente iteración, donde se podrán priorizar apropiadamente.

- El equipo de developers, desarrollan las especificaciones instrumentadas, crean pruebas de fallos y luego implementan la funcionalidad, utilizando el artefacto de historias de usuario con sus especificaciones o criterios de aceptación y también el fixture usando Frameworks como el Fitnesse y Cucumber para la automatización de las pruebas.

- Demostrar. Una vez que todas las pruebas pasan, la historia puede ser demostrada al dueño del producto.

5.3.3 TDD (Test Driven Development) paso a paso

A continuación se describe la aplicación de la técnica TDD en varios pasos. (Beck, Test-Driven Development By Example, 2002)

- Write a test (Escriba una Prueba): En este paso se escribe el requisito en forma de Test. Se pueden utilizar frameworks de testing como JUnit para Java, NUnit para .NET, PYunit para Python, PHPUnit para PHP entre otros.
- Se deben escribir los Test Unitarios por cada método que se ha realizado para los criterios de aceptación.
- Test de Aceptación: donde se asegura que se ha desarrollado la funcionalidad adecuada.
- Run tests see new failure (Verificar si la prueba falla): Verificar si la prueba falla, esto garantiza la calidad de la implementación, ya que en el caso de que la prueba falla, se volverá a escribir la prueba, volviendo al primer paso.
- Write some code (Escribir el código): Una vez que se tienen los test, se procede a escribir el código estrictamente necesario para que se cumpla el requerimiento donde se hace referencia a la expresión “Keep It Simple, Stupid!” (Déjelo Simple), se debe ignorar si el código se ve feo. Si bien se vendrán dudas al momento de implementar el código referente al comportamiento referente a ciertas variables de entrada y salida, esto se debe escribir para tenerlo en cuenta en una próxima iteración, más no se debe desarrollar en el momento.
- Run tests see all pass (Ejecutar la prueba y verificar que funciona): Verificar si la prueba funciona correctamente con el código escrito en el paso anterior. Se puede usar herramientas como el Selenium ya que es un entorno de pruebas para aplicaciones basadas en web que facilitara la ejecución del código implementado.
- Refactor (Refactorización): En este paso se realizará la refactorización del código, lo cual será la modificación del diseño sin alterar su comportamiento. Se buscan las líneas duplicadas y se eliminan, aplicando la refactorización.
- Es importante revisar patrones de diseño, una técnica útil a implementar es S.O.L.I.D que son cinco principios básicos de la programación orientada a objetos y el diseño. Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de mantener y ampliar con el tiempo.

5.3.4 SCRUM