DESARROLLO DIRIGIDO POR TEST (TDD) UTILIZANDO EL FRAMEWORK JUNIT EN UN SISTEMA WEB DE ASIGNACIÓN DE AULAS DE LOS LABORATORIOS GENERALES DE COMPUTACIÓN DE LA ESPE, APLICANDO LA METODOLOGÍA AGILE UNIFIED PROCESS (AUP)

Tatiana Pozo Molina[1], Carlos Aucancela Maguana[2]
Ing. Cecilia Hinojosa[3] Ing. Aly Abdelrahaman[4]

[1]Escuela Politécnica del Ejército. Ecuador, pandy1517@hotmail.com
[2]Escuela Politécnica del Ejército. Ecuador, carlos_javier_18@hotmail.com
[3]Escuela Politécnica del Ejército. Ecuador, chinojosa@espe.edu.ec
[4]Escuela Politécnica del Ejército. Ecuador, abdito_8@hotmail.com

RESUMEN

El presente trabajo está orientado a realizar un estudio del desarrollo guiado por pruebas, o Test Driven Development (TDD) una técnica de programación que involucra principalmente dos prácticas: escribir las pruebas primero (Test First Development) y refactorización (Refactoring), aplicado al desarrollo de una herramienta informática que permita minimizar la gestión de reservas de los laboratorios computacionales de la ESPE.

El procedimiento de esta técnica empieza por escribir una prueba acorde a un requerimiento específico, en la cual se verifica que dichas pruebas fallen, luego se implementa el mínimo código que haga que la prueba pase satisfactoriamente para proceder a refactorizar el código escrito si fuere necesario.

La aplicación práctica de TDD fue realizada en el desarrollo de un sistema web de reserva de laboratorios computacionales de la ESPE (SILVERLAB), siguiendo los lineamientos de la metodología de desarrollo de software AUP, la cual se basa en cortas iteraciones, desde el levantamiento de requisitos, análisis, diseño e implementación. Para las pruebas se utilizó la herramienta JUNIT con la finalidad de verificar, manejar y ejecutar conjuntos de pruebas automatizadas.

El producto software resultante es un sistema web distribuido en lenguaje de programación JAVA, basado en el patrón de diseño Modelo Vista Controlador que permite separar en componentes dicho sistema, además posee un motor de base de datos MYSQL.

Tras el desarrollo de la aplicación se evidenció las ventajas que provee TDD tales como:

- ✓ *Permitir identificar lo que es realmente imprescindible implementar por lo que se ahorrará tiempo desarrollando código que luego no se usará.*
- ✓ *Ayuda al programador a tener un mayor nivel de confianza en el código desarrollado.*
- ✓ *Fuerza a un estricto análisis y diseño, ya que el desarrollador no puede crear código de producción sin entender realmente cuales deberían ser los resultado deseados y como probarlos.*
- ✓ *El conjunto de test unitarios proporciona constante retroalimentación de que cada uno de los componentes sigue funcionando.*

El aporte de este estudio ofrece más que una simple validación del cumplimiento de requisitos, ya que sirvió como guía al diseño de la aplicación práctica haciendo que el código que funciona retroalimente las nuevas decisiones de diseño.

Palabras Clave: TDD, Refactorizar, JUnit, UnitTest.

ABSTRACT

This paper aims to conduct a study of Test Driven Development (TDD) it`s a programming technique that mainly involves two practices: Writing the first test and code refactoring applied to the development of a software tool that allows to minimize the management computer labs of the ESPE.

The procedure of this technique begins with a writing of a test according to a specific requirement, which verifies that these tests fail, and then write the minimum code that successfully pass the test and then proceed to refactor the code written.

The practical application of TDD was made in the development of a ESPE web booking computational laboratories (SILVERLAB), following the guidelines of the software development methodology AUP, which is based on short iterations, from the lifting requirements, analysis, design and implementation. For testing tool was used Junit in order to verify, manage and execute automated test suites.

The resulting software product is a distributed web system in JAVA programming language, based on the standard Model View Controller design for separating the system components, also it has a database engine MySQL.

After the development of the application was demonstrated the advantages of TDD such as:

- ✓ Allow identify what is really essential to implement so it will save time developing code then not be used.
- ✓ Help the developer to have a higher level of confidence in the code developed.
- ✓ *F*orces a strict analysis and design, as the developer cannot create production code without really understanding what should be the test result as desired.
- ✓ The test set unit provides constant feedback from each of the components is still running.

The contribution of this study offers more than simple compliance validation requirements, because guided us for the design of practical application making the code that runs feeding into new-design decisions.

Keywords: TDD, Refactoring, JUnit, UnitTest.

1. INTRODUCCIÓN

El proceso de desarrollo de software de calidad se ve opacado cuando no existe el suficiente diálogo o un inadecuado entendimiento de las necesidades de los usuarios, ya que la fase de pruebas es realizada después de la etapa de desarrollo, implicando grandes esfuerzos al momento de realizar cambios en los requisitos.

El desarrollo dirigido por pruebas (TDD) es una técnica que se acopla a las metodologías ágiles, destacando su sencillez y confiabilidad; TDD pretende guiar el desarrollo mediante la creación continua de tests automatizados y en base a los mismos construir la solución que los satisfaga, lo cual reduce los costos de implementación en un equipo de desarrollo, llegando a obtener un eficiente diseño de software con la ayuda de las pruebas unitarias.

La Escuela Politécnica del Ejército se ha visto en la necesidad de crear un sistema web para la gestión de reservas de los laboratorios generales de computación, con el fin de optimizar dicho proceso, el cual se lo venía haciendo de forma manual. Por lo que el objetivo del presente proyecto es atender esta necesidad aplicando TDD en dicho sistema, para lograr que el resultado del desarrollo sea “código limpio que funcione”, es decir obtener un código de calidad sin duplicación y de correcta funcionalidad, generados a través de los resultados de los test.

2. METODOLOGÍA

El Desarrollo Dirigido por Test se basa en la ejecución de pruebas antes de la etapa de implementación, por lo que para su comprobación se aplicó dicha técnica en el desarrollo de un sistema para el control de la asignación de laboratorios generales de computación de la Escuela Politécnica del Ejército, el proyecto se encuentra orientado a la web basado en la metodología ágil de desarrollo de software AUP, la cual se adapta a un modelo iterativo e incremental (entregas pequeñas de software, con ciclos rápidos), cooperativo (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), sencillo (el método en sí mismo es fácil de aprender y modificar), y adaptable (permite realizar cambios de último momento)[8], por lo que cada iteración es una versión del sistema mejorado y en donde el usuario forma parte importante durante toda la fase de desarrollo del proyecto con la finalidad de ir puliendo errores de las pruebas generadas.

AUP está basado en disciplinas y entregables incrementales con el tiempo. El ciclo de vida en proyectos grandes es serial mientras que en los pequeños es iterativo.

Las disciplinas se desarrollan de una manera iterativa, definiendo las actividades las cuales los miembros del equipo de desarrollo construirán, validarán y entregaran el software el cual cumple las necesidades de los clientes. Las disciplinas son:

- ✓ Modelado
- ✓ Implementación
- ✓ Prueba
- ✓ Despliegue
- ✓ Administración de la configuración
- ✓ Administración de Proyecto
- ✓ Entorno

AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (Test Driven Development - TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad.

3. HERRAMIENTAS

El presente proyecto utiliza las siguientes herramientas para el desarrollo del sistema web **Ver Tabla 1**.

Tabla 1: Herramientas de desarrollo y pruebas

Herramienta	Utilidad
Netbeans	Herramienta para el desarrollo del código fuente de la aplicación.
JEE	Plataforma de programación web para el desarrollo y ejecutar software de aplicaciones en n capas.
MySQL	Sistema gestor de base de datos relacional, multihilo y multiusuario.
JUnit	JUnit es un conjunto de bibliotecas (framework), creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. [1]
Jsp	JSP (Java Server Pages) es una tecnología web, del lado del servidor, están compuestas de código HTML/XML mezclado con etiquetas esenciales para programar scripts de servidor en sintaxis Java. [2]
Jsf	JavaServer Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. [3]
Pmd	PMD es una herramienta que comprueba que nuestra aplicación cumpla una serie de reglas que nos ayudan a obtener un código más elegante, sencillo y mantenible. [4]
SourceForge	SourceForge es una central de desarrollos de software que controla y gestiona varios proyectos de software libre y actúa como un repositorio de código fuente. [5]

Tests Unitarios

Para comprender el funcionamiento de TDD es necesario explicar las características que deben tener los tests unitarios generados durante la aplicación de esta metodología. Los Tests generados durante los ciclos de desarrollo utilizando TDD no son los tests ejecutados comúnmente durante el desarrollo clásico en el cual los desarrolladores prueban los cambios y/o funcionalidades nuevas del producto ejecutando el programa y verificando que las cosas "funcionen" según lo deseado [6], sino que la finalidad de realizar test unitarios es encontrar una solución óptima a un determinado requerimiento antes de implementar código.

TDD no implica solamente **realizar** pruebas, sino implica **tener** Tests automatizados, fácilmente repetibles y que permitan ser ejecutados con el mínimo esfuerzo posible. Estos tests automatizados no son tests del tipo de caja negra sobre el producto o sobre módulos del producto, sino tests aplicados directamente sobre las unidades mínimas de comportamiento que conforman el producto como pueden ser por ejemplo los métodos de una clase en un entorno de objetos con clases. [7]

4. FUNDAMENTOS TEÓRICOS.

TDD es una técnica para diseñar software que se centra en tres pilares fundamentales.

- ✓ 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.

TDD se basa en el siguiente ciclo de desarrollo basado en el libro de *Ensayos de desarrollo impulsado por ejemplo* [9]:

1. **Elegir un requerimiento:** Consiste en que el desarrollador escribe un conjunto de requisitos que crea que dará mayor conocimiento del problema y que a la vez sea fácil de implementar.

2. **Escribir la prueba**. Para escribir la prueba, el desarrollador debe entender claramente las especificaciones y los requisitos. El diseño del documento deberá cubrir todos los escenarios de prueba y condición de excepciones.

3. **Escribir el código haciendo que pase la prueba**. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces. Ésta es la parte conducida por el diseño, del TDD. Como parte de la calibración de la prueba, el código debe fallar la prueba significativamente las primeras veces.

4. **Ejecutar las pruebas automatizadas**. Si pasan, el programador puede garantizar que el código resuelve los casos de prueba escritos. Si hay fallos, el código no resolvió los casos de prueba.

5. **Refactorización y limpieza en el código**. Después se vuelven a efectuar los casos de prueba y se observan los resultados.

6. **Repetición**. Después se repetirá el ciclo y se comenzará a agregar las funcionalidades adicionales o a arreglar cualquier error.

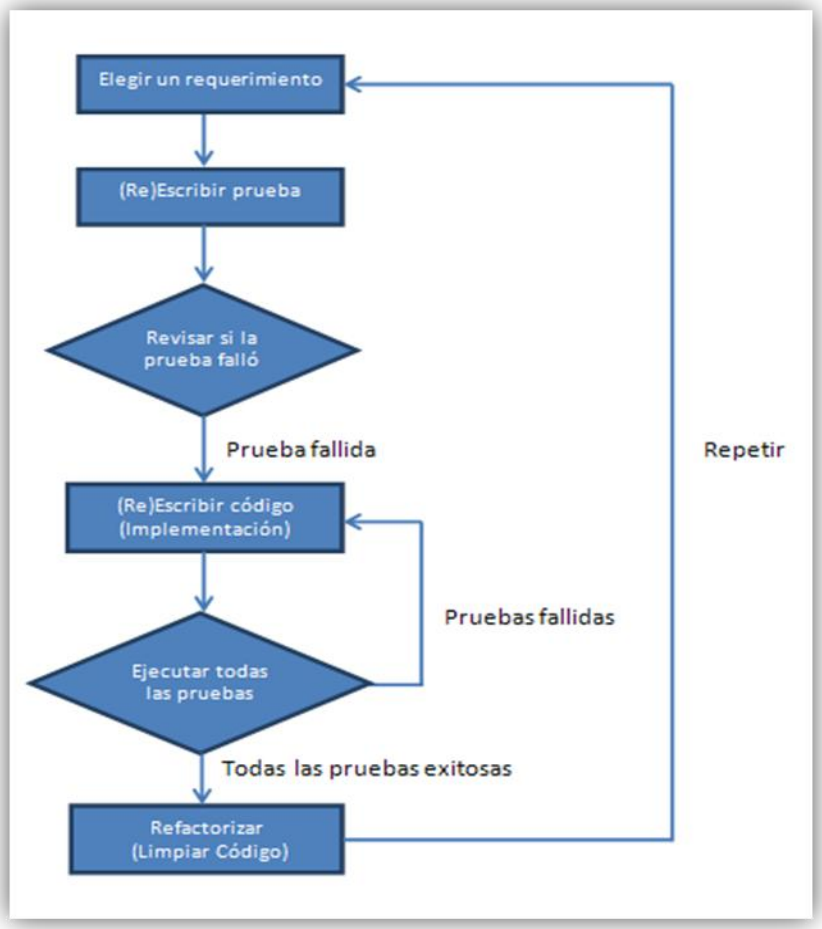

Fig. 1: Ciclo de Desarrollo de TDD

Behavior Driven Development (BDD)

BDD reorienta el enfoque al comportamiento del sistema. BDD usa una plantilla para poder pensar en el comportamiento de las pruebas del código [10]:

Dado (Given)**,** un contexto inicial
Cuando (When) un evento se produce
Entonces (Then) asegure algunos resultados.

Acceptance Test Driven Development (ATDD)

Ayuda a coordinar los proyectos de software de forma de entregar lo que el cliente desea.
Las pruebas de aceptación son especificaciones de comportamiento y funcionalidad deseados para un sistema. ATDD determina el cómo se implementará una determinada historia de usuario o caso de uso, el sistema trata determinadas condiciones y entradas. Una buena prueba de aceptación debe ser [11]:

- ✓ Propiedad de los clientes
- ✓ Escrito en conjunto con los clientes, desarrolladores y analistas de prueba
- ✓ Sobre el Qué y no sobre el Cómo
- ✓ Expresada en lenguaje de dominio del problema
- ✓ Conciso, preciso y sin ambigüedades

5. APLICACIÓN UTILIZANDO TDD

Con la finalidad de aplicar el Desarrollo Dirigido por Test, se desarrolló un sistema orientado a la web para la gestión de reservas de los laboratorios computacionales de la Escuela Politécnica del Ejército.

La herramienta es un sistema web distribuido, desarrollado en el lenguaje de programación JAVA, basado en el patrón de diseño Modelo Vista Controlador que permite separar en componentes dicho sistema, además posee un motor de base de datos MYSQL.

Para las pruebas se utilizó la herramienta JUNIT con la finalidad de verificar, manejar y ejecutar conjuntos de pruebas automatizadas.

Entre las principales características del sistema desarrollado se puede mencionar:

- ✓ Administración de usuarios y perfiles de acceso en el sistema.
- ✓ Configuración de periodos académicos.
- ✓ Disponibilidad de laboratorios en horas detalladas.
- ✓ Creación de reservas.
- ✓ Gestión de entidades (laboratorios, materias, carreras)

Fig. 2. Pantalla inicial del sistema SILVERLAB

A continuación se presenta la pantalla principal del sistema SILVERLAB, donde muestra las principales entidades que abarca el sistema cada una tiene submenús propios de la entidad.

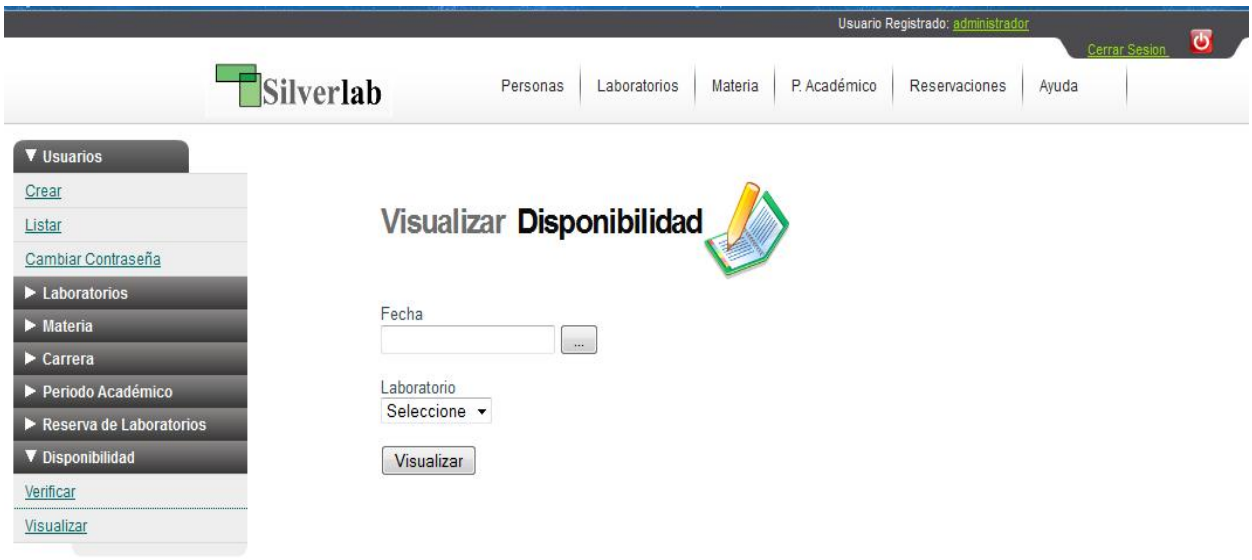

Fig. 3. Pantalla inicial del sistema SILVERLAB

6. RESULTADOS

Los resultados obtenidos tras el estudio de la técnica TDD, se realizaron mediante el análisis y estudio de técnicas que también se basan en la filosofía de TDD como son ATDD, BDD explicadas anteriormente.

La siguiente tabla muestra una comparación entre cada una de ellas clasificadas por criterios de evaluación y prioridades.

CRITERIO	
1 = Más Bajo	10 = Más Alto

PESO	
1% = Más Bajo	10% = Más Alto

TABLA 2: COMPARACIÓN ENTRE TDD – ATDD – BDD MEDIANTE CRITERIO DE EVALUACIÓN

Nro.	Criterio de Evaluación	TÉCNICAS DE DESARROLLO				
		TDD	ATDD	BDD	Peso	General
1	Lenguaje simple y natural	5	5	7	7%	10
2	Multilenguaje	9	7	8	5%	10
3	Consistencia	9	8	7	20%	10
4	Porcentaje de utilidad	10	8	8	10%	10
5	Facilidad al aplicar	8	9	7	7%	10
6	Ayuda y documentación	8	8	8	4%	10
7	Incremento de tiempo de desarrollo	7	6	6	7%	10
8	Porcentaje de disminución de defectos	9	8	7	20%	10
9	Cantidad Documentación en ingles	9	9	7	4%	10
10	Cantidad Documentación en español	8	7	5	4%	10
11	Conocimientos de programación	9	7	6	10%	10
12	Utilización de software libre	10	10	10	2%	10
	TOTAL	101	92	86	100	130
	TOTAL PORCENTAJE	77.69%	70.76%	66.15%	100%	100%
	TOTAL PORCENTAJE PESO	80.55	70.61%	70%	100%	

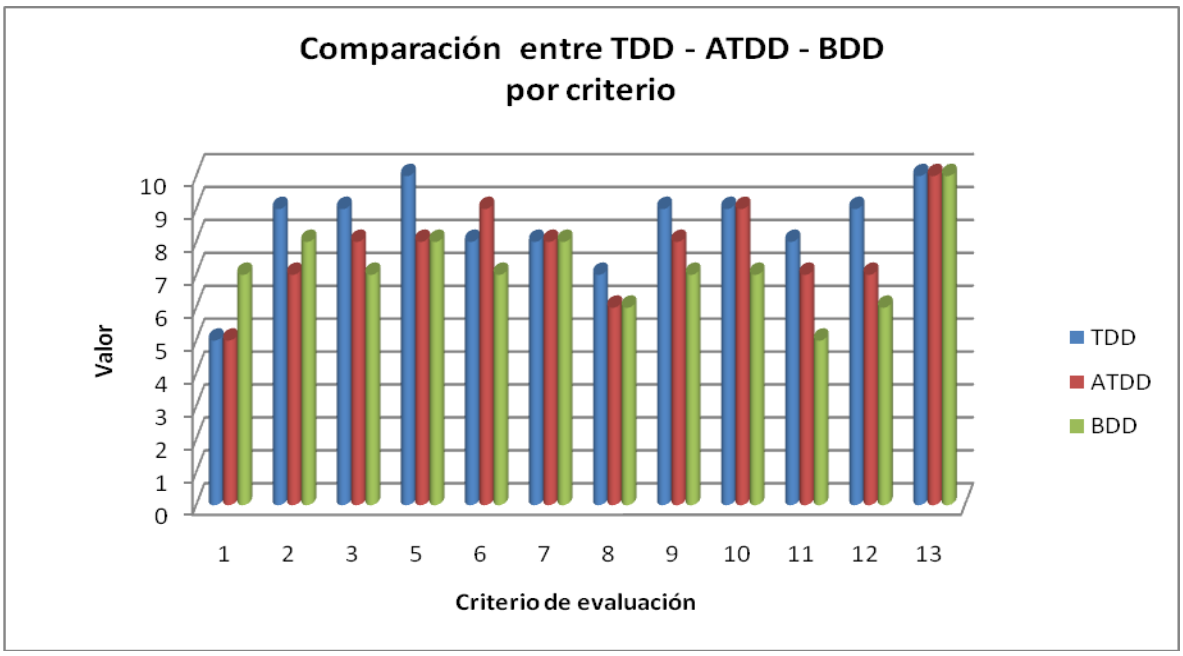

FIG. 4. COMPARACIÓN ENTRE TDD - ATDD – BDD POR CRITERIO DE EVALUACIÓN

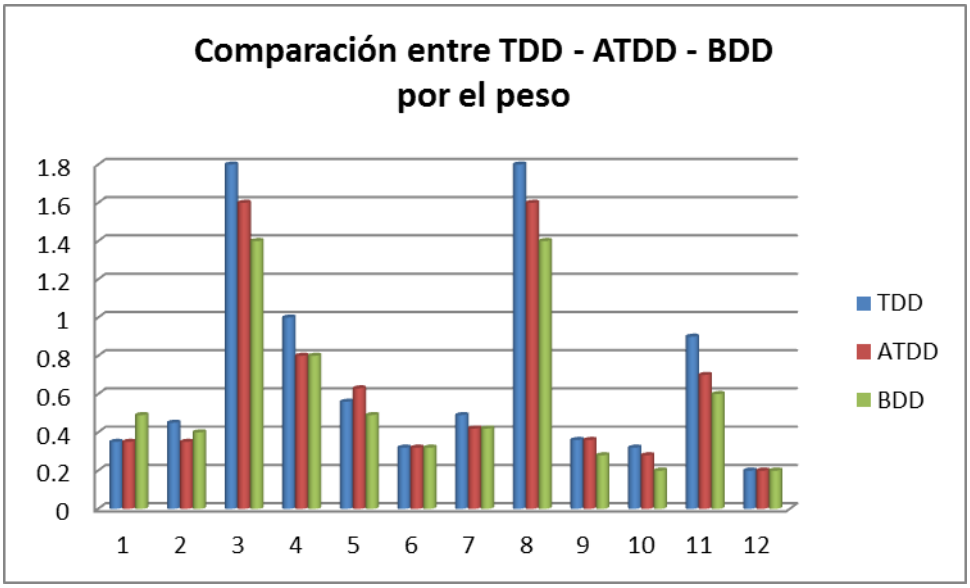

FIG. 3. COMPARACIÓN ENTRE TDD - ATDD – BDD POR PESO

7. TRABAJOS RELACIONADOS

En cuanto al estudio del Desarrollo Dirigido por Test TDD, existen algunas técnicas que también han llevado la filosofía de TDD, estos son Acceptance Test Driven Development (ATDD) y Behaviour Driven Development (BDD).

ATDD [12] es una evolución de TDD, orientada a resolver la falta de comunicación con el cliente y abordar adecuadamente los cambios que pueda introducir a lo largo del proceso de desarrollo. En ATDD el proceso de desarrollo está dirigido por las pruebas de aceptación, las cuales deberían ser escritas por el cliente con la ayuda de un desarrollador. Un requisito en ATDD es una Historia de Usuario, especificada con una breve descripción narrativa y a la cual se le asocia un conjunto de pruebas de aceptación, escritas normalmente en lenguaje natural como lo detalla Roger en [13].

Por otra parte, BDD surge como otra propuesta promovida por North [14], con el propósito de conectar las pruebas unitarias con los requisitos.
BDD es un enfoque que empieza por identificar la funcionalidad del negocio, y después profundiza en el conjunto de requisitos que esta funcionalidad alcanzará.
Ambos, ATDD y BDD, son muy similares en cuanto a que están orientados a la automatización de pruebas y generación de código (al menos parcial).
En BDD y ATDD las pruebas se automatizan inmediatamente al ser identificadas, no existe separación temporal en cuanto a la especificación de la prueba y su correspondiente implementación.

En cuanto al desarrollo de sistemas utilizando la técnica TDD existen algunos proyectos relacionados entre ellos:

FITNESSE

FitNesse es un proyecto open Source (http://fitnesse.org/FitNesse), que permite a los clientes, testers y programadores diseñar y entender lo que sus programas deben hacer, y comparar automáticamente con lo que realmente hacen. La finalidad es comparar las expectativas de los clientes y si estas se cumplen con los resultados obtenidos.

Técnicamente FitNesse tiene alrededor de 50 mil líneas de código java. El mismo se encuentra construido en base al Desarrollo Dirigido por Test (TDD). El sistema es muy robusto y se encuentra abierto para la comunidad donde se puede aportar con el desarrollo de nuevas versiones.

DIMDWARF APPLICATION SERVER

Es un proyecto open Source disponible para la comunidad en http://dimdwarf.sourceforge.net/. Esta desarrollado por completo con TDD / BDD y tratando de escribir el código más limpio posible (sólido, limpio, de calidad, etc.).

Dimdwarf es un servidor de aplicaciones distribuidas para plataforma Java. Donde se escribe un único subproceso, el código de POJO por eventos - el servidor hace que sea multi-hilo, persistente y transaccional. Los usuarios finales de la aplicación son los desarrolladores de juegos multijugador en línea.

"Para aportar cambios al proyecto se deberá seguir las siguientes guías:

1. Aplicar la metodología ágil Extreme Programming (XP).

- *Ejecutar todas las pruebas*
- *No contener duplicación*
- *Desarrollar únicamente el requisito establecido*
- *Reducir al mínimo el número de clases y métodos*

2. Las pruebas deben estar escritas justo antes del código de producción (es decir, usar TDD)
3. Las pruebas deben estar organizadas como especificaciones ejecutables de comportamiento del sistema (es decir, el uso de BDD)
4. Sigue el buen diseño orientado a objetos, por ejemplo mediante la aplicación de los principios 'SÓLID"[15].

8. CONCLUSIONES Y TRABAJO FUTURO

Tras el estudio de la técnica del Desarrollo Dirigido por Test, se ha podido constatar que la misma no es solamente una técnica que abarca el testing de la aplicación, sino que es una técnica de diseño, ya que intenta mejorar el enfoque de desarrollo obteniendo código de calidad.

Se realizó una comparación entre algunas técnicas que también han llevado la filosofía de TDD, como son Acceptance Test Driven Development (ATDD) y Behaviour Driven Development (BDD), ambas son muy similares en cuanto a que están orientados a la automatización de pruebas y generación de código, pero la diferencia radica en que TDD busca que el programador vaya más allá que implementar código, sino que se convierta en un diseñador de software.

El acoplarse al TDD no es una tarea sencilla, requiere de un largo proceso de adaptabilidad a buenas prácticas de desarrollo ágil, en principio el uso del mismo se torna complejo necesitando de mayor esfuerzo y tiempo para poder lograr resultados eficientes.

El Desarrollo Dirigido por Test (TDD) se acopla a diferentes lenguajes de programación como son Ruby, PHP, Java, Pearl, entre otros. Para la aplicación desarrollada se utilizó la plataforma de programación Java Enterprise Edition, facilitando el desarrollo de una aplicación distribuida en niveles de programación, permitiendo ejecutar funcionalidades sencillas que se acoplan al requerimiento del usuario.

La metodología AUP aplica técnicas ágiles como el Desarrollo Dirigido por Pruebas (Test Driven Development - TDD), Modelado Ágil y Gestión de Cambios Ágil, por lo que se acopó de manera significativa en el desarrollo del caso práctico, permitiendo tener una correcta distribución de las actividades de trabajo y a la vez centrarse en actividades de alto valor, logrando una aplicación distribuida y sencilla.

Adicionalmente se utilizó la arquitectura MVC, logrando una clara separación entre interfaz, lógica de negocio y de presentación, facilitando la realización de pruebas unitarias de sus clases, obteniendo un código flexible para cambios.

La aplicación desarrollada fue publicada en el repositorio web sourceforge.net, el cual se encuentra a disposición de la comunidad con la finalidad de aportar, comentar, y continuar con el desarrollo de funcionalidades sobre la aplicación.

Cabe mencionar que cuando existen proyectos realizados sin TDD, se puede realizar test de regresión que consiste en la aplicación de test en el front-end (primera capa), con la finalidad de encontrar divergencias funcionales respecto al comportamiento esperado del software.

Como trabajo futuro se pretende aplicar ATDD al sistema desarrollado con la finalidad de garantizar un óptimo funcionamiento que abarque comportamientos y test de aceptación.

9. RECOMENDACIONES

Para obtener un mejor resultado en cuanto al uso de la técnica TDD, se recomienda:

- ✓ Trabajar conjuntamente con buenas prácticas del desarrollo ágil como por ejemplo la utilización de patrones de diseño, buen uso de semántica, integración continua, entre otros.

- ✓ Intentar crear los tests antes de la implementación, ya que si se realizan los mismos después de la implementación se está cayendo en el desarrollo tradicional, por lo que se pierde todas las ventajas que aporta usar TDD.

- ✓ Crear un test por iteración y solo implementar el mínimo código necesario para resolver ese caso. No es bueno “emocionarse” implementando y desarrollar más de lo necesario para resolver el caso de prueba, ya que si se desarrolla más casos se pierde una gran parte de la eficacia de esta técnica.

- ✓ No intentar automatizar todo el proceso de prueba, puede no ser viable ni práctico. La prueba debe ser estratégica en la búsqueda de defectos.

- ✓ Realizar un desarrollo de parejas durante la aplicación del TDD, donde el equipo debe tener un nivel promedio de experiencia en el desarrollo de software y haber aprendido de errores pasados para de esta manera obtener mejores resultados.

- ✓ Cuando se realice un desarrollo de gran dimensión es importante dividir por módulos de funcionalidad, ya que el proceso de testing se torna complejo y es necesaria la aplicación de herramientas adicionales de pruebas como Jmocks y Stubs.

En cuanto a la aplicación desarrollada, el administrador encargado deberá tener un amplio conocimiento en cuanto a su funcionalidad, para ayudar de soporte a otros usuarios.

El código estará disponible en el repositorio web de SourceForge con la finalidad de poder acceder a realizar mejoras a la aplicación, por lo que se recomienda que cualquier cambio deba ser realizado de forma transparente para el usuario final.

10. REFERENCIAS BIBLIOGRÁFICAS

[1] Anónimo. (s.f.). *JUnit.org*. Recuperado el 23 de Marzo de 2011, de JUnit.org: http://www.junit.org/

[2] Alvarez, M. A. (08 de Julio de 2002). *desarrolloweb.com Que es JSP*. Recuperado el 02 de Abril de 2011, de desarrolloweb.com Que es JSP: http://www.desarrolloweb.com/articulos/831.php

[3] Anónimo. (18 de Mayo de 2006). *Wikipedia JavaServer Faces* . Recuperado el 07 de Abril de 2011, de Wikipedia JavaServer Faces : http://es.wikipedia.org/wiki/JavaServer_Faces

[4] http://mundogeek.net/archivos/2009/03/08/mejora-tu-codigo-java-con-pmd/

[5] Anónimo. (2005). *Wikipedia SourceForge* . Recuperado el 20 de Junio de 2011, de Wikipedia SourceForge : http://es.wikipedia.org/wiki/SourceForge

[6] Test-driven development: by example Escrito por Kent Beck Addison Wesley, 2003

[7] Diseño ágil con TDD (Carlos Blé Jurado)

[8] Abrahamsson, P., Salo, O., Ronkainen, J., Agile Software Development Methods. Review and Analysis, VTT, 2002.

[9] Beck, K. (2003). *Test Driven Development: by Example.*

[10][11] Gómez, D. (26 de Marzo de 2009). *www.dos¡deas.com*. Obtenido de www.dos¡deas.com: http://www.dosideas.com/noticias/metodologias/484-calidad-de-software-con-bdd-y-atdd.html

[12] Beck, K. *Test Driven Development: By Example*. Addison Wesley, 2003

[13] Roger, S. *Acceptance testing vs. unit testing: A developer's perspective*. Lecture Notes in Computer Science, 2004 – Springer

[14] North, D. *Introducing BDD*, Better Software, March, 2006.

[15] Esko Luontola , Dimdwarf Application Server http://dimdwarf.sourceforge.net/