Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), mayo-junio 2024,
Volumen 8, Número 3.

https://doi.org/10.37811/cl_rcm.v8i3

IMPLEMENTACIÓN DE LA METODOLOGÍA TEST DRIVEN DEVELOPMENT EN EL DESARROLLO DE UN PROTOTIPO DE SISTEMA DE CONTROL DE INVENTARIO

IMPLEMENTATION OF THE TEST DRIVEN DEVELOPMENT METHODOLOGY IN THE DEVELOPMENT OF A PROTOTYPE INVENTORY CONTROL SYSTEM

Angel Geovanny Cudco Pomagualli
Universidad de las Fuerzas Armadas, Ecuador

DOI: https://doi.org/10.37811/cl_rcm.v8i3.11552

Implementación de la Metodología Test Driven Development en el Desarrollo de un Prototipo de Sistema de Control de Inventario

Angel Geovanny Cudco Pomagualli[1]
agcudco@espe.edu.ec
https://orcid.org/0000-0003-4825-650X
Universidad de las Fuerzas Armadas – ESPE
Ecuador

RESUMEN

Actualmente, los sistemas de información están presentes en todos los aspectos de nuestra vida, y las pequeñas y medianas empresas no son la excepción. Sin embargo, muchas aún carecen de un sistema eficiente y confiable para administrar su información. Este trabajo se centra en la metodología Test Driven Development (TDD), con el objetivo de identificar sus ventajas en la construcción de sistemas. TDD es una metodología de desarrollo de software basada en pruebas para guiar el proceso. Los resultados muestran que TDD, integrada en el framework JUnit, no solo cubre el testing de la aplicación, sino que también promueve un diseño óptimo y eficiente. Esto mejora la codificación del software, resultando en un código más tolerante al cambio, robusto y seguro. Tras analizar las ventajas de TDD y los principios de Scrum, se desarrolló e implementó un prototipo de sistema de control de inventario utilizando Java SE y MySQL. Este prototipo permitirá optimizar los procesos de compra, venta y control de productos de cualquier PYME.

Palabras clave: sistemas de información, test driven development (TDD), desarrollo de software, junit, SCRUM

[1] Autor principal.
Correspondencia: agcudco@espe.edu.ec

Implementation of the Test Driven Development Methodology in the development of a prototype Inventory Control System

ABSTRACT

Nowaday, information systems are present in all aspects of our lives, and small and medium-sized enterprises are no exception. However, many still lack an efficient and reliable system to manage their information. This work focuses on the Test Driven Development (TDD) methodology, with the aim of identifying its advantages in system construction. TDD is a software development methodology based on testing to guide the process. The results show that TDD, integrated into the JUnit framework, not only covers application testing but also promotes optimal and efficient design. This improves software coding, resulting in code that is more tolerant to change, robust, and secure. After analyzing the advantages of TDD and the principles of Scrum, a prototype inventory control system was developed and implemented using Java SE and MySQL. This prototype will optimize the processes of purchasing, selling, and controlling products for any PYME.

***Keywords*:** information systems, test driven development (TDD), software development, Junit, SCRUM

Artículo recibido 20 abril 2024

Aceptado para publicación: 25 mayo 2024

INTRODUCCIÓN

Garantizar la calidad de los productos de software y mejorar la productividad a lo largo de todo el ciclo de vida, desde el inicio del desarrollo hasta las etapas de mantenimiento, representa uno de los desafíos más significativos. En la actualidad, los requisitos son más volátiles que nunca, lo que potencialmente conlleva la introducción de numerosos errores. Estos errores pueden causar que las funcionalidades previamente implementadas comiencen a fallar y que aparezcan numerosos "bugs" durante la fase de mantenimiento (Vaca et al., 2014).

A menudo, los equipos de desarrollo de proyectos de software centran su atención en la arquitectura del software y en la adopción de nuevas tecnologías, dejando de lado la importancia de una metodología de trabajo adecuada. Tanto las metodologías tradicionales como las ágiles tienen el propósito de guiar al desarrollador desde la fase inicial hasta la fase de mantenimiento.

El Desarrollo Guiado por Pruebas (TDD), una técnica fundamental en el ciclo de desarrollo, permite a los desarrolladores analizar, modificar y refactorizar el código fuente sin el temor de afectar las funcionalidades existentes, ya que cada una de ellas está asociada a una prueba que debe superarse.

En este trabajo, se llevó a cabo la implementación de un prototipo de sistema para el control de inventario de cualquier PYME utilizando la metodología TDD. Esta metodología destaca por escribir la menor cantidad posible de código para obtener el resultado deseado, y fue implementada a través del framework JUnit.

METODOLOGÍA

El trabajo se llevó a cabo desde un enfoque cuantitativo y con un diseño de tipo cuasi experimental. Su principal foco es la aplicación de la metodología Test Driven Development (TDD) dentro del marco de trabajo Scrum, para evaluar la calidad del código en las operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Se logró que el código tenga una legibilidad adecuada, esté libre de duplicaciones y redundancias, y mantenga un alto nivel de mantenibilidad. El objetivo principal es lograr un código limpio y funcional que se integre de manera efectiva en el contexto ágil de desarrollo proporcionado por Scrum.

Para el desarrollo del prototipo se tomaron como referencia un ciber, un tienda de ropa y mini-mercado, y se establecieron los siguientes roles:

Tabla 1. Roles del sistema.

Rol	Descripción	Responsabilidades
Administrador	Persona responsable de administrar sistema.	Gestiona el personal de la empresa, compras, ventas, administrar el almacén, realizar el control de productos y generar informes, reportes gráficos y en formato pdf.
Vendedor	Persona que se encarga del área de ventas.	Gestionar las ventas.

Fuente: Autor

Finalizada la especificación de requisitos de acuerdo a su prioridad y considerando la complejidad de cada uno, se procedió a transformarlos en historias de usuario. Tras las reuniones con los voluntarios y el análisis de los resultados, se identificaron treinta y siete requisitos, que se desglosan en ocho historias técnicas y veintinueve historias de usuario. La siguiente tabla detalla las historias de usuario e historias técnicas utilizadas en el desarrollo del sistema.

Tabla 2. Historias de usuario y técnicas

Código	Descripción	Esfuerzo
HT-01	Establecer los requerimientos del sistema	18
HT-02	Establecer la arquitectura del sistema	18
HT-03	Diseño de Estándar de Codificación	6
HT-04	Diseño del estándar de interfaz	18
HT-05	Realizar la Base Datos (Modelo entidad relación, lógico, físico, script diccionario de datos)	12
HU-01	Ingresar categoría	12
HU-02	Modificar categoría	12
HU-03	Agregar vigencia de categoría	12
HU-04	Gestión productos	12
HU-05	Agregar productos	12
HU-06	Modificar productos	12
HU-07	Buscar para el módulo almacén	12
HU-08	Establecer vigencia de producto	12
HU-09	Reportes para módulo almacén	12
HU-10	Crear sesión de usuarios	18
HU-11	Gestionar roles	12
HU-12	Asignar funcionalidad de acuerdo con el rol de usuario	12

HU-14	Buscar y visualizar usuarios empleados de la empresa	6
HU-15	Gestionar proveedores	12
HU-16	Gestionar compras a proveedores	12
HU-17	Consultar compras por fecha, mes	12
HU-18	Buscar proveedores	6
HU-19	Historial de precios	6
HU-20	Reportes para el módulo proveedores	12
HU-21	Gestionar clientes	12
HU-22	Gestionar venta	24
HU-23	Consultar ventas por día, fecha, mes	12
HU-24	Reportes para el módulo ventas	12
HU-25	Administrar caja	24
Hu-26	Historial de caja	12
HU-27	Generar Inventario	24
HU-28	Resumen de saldos y movimiento de productos.	30
HU-29	Graficas estadísticas de compras y ventas.	30
HT-07	Capacitación de usuarios	18
HT-08	Documentación final del Sistema	24

Posteriormente se estableció el Sprint Backlog.

Tabla 3. Sprint Backlog

ID	Descripción	Esfuerzo
SPRINT 1		**60**
HT-01	Establecer los requerimientos del sistema	18
HT-02	Establecer la arquitectura del sistema	18
HT-03	Diseño de Estándar de Codificación	6
HT-04	Diseño del estándar de interfaz	18
SPRINT 2		**60**
HT-05	Realizar la Base Datos (Modelo entidad relación, lógico, físico, script diccionario de datos)	12
HU-01	Ingresar categoría	12
HU-02	Modificar categoría	12
HU-03	Agregar vigencia de categoría	12
HU-04	Gestión productos	12

ID	Descripción	Esfuerzo
SPRINT 3		**60**
HU-05	Agregar productos	12
HU-06	Modificar productos	12
HU-07	Buscar para el módulo almacén	12
HU-08	Establecer vigencia de producto	12
HU-09	Reportes para modulo almacén	12
SPRINT 4		**60**
HU-10	Crear sesión de usuarios	18
HU-11	Gestionar roles	12
HU-12	Asignar funcionalidad de acuerdo con el rol de usuario	12
HU-13	Activar estado de vigencia de la funcionalidad asignada	12
HU-14	Buscar y visualizar usuarios empleados de la empresa	6
SPRINT 5		**60**
HU-15	Gestionar proveedores	12
HU-16	Gestionar compras a proveedores.	12
HU-17	Consultar compras por fecha, mes.	12
HU-18	Buscar proveedores	6
HU-19	Historial de precios	6
HU-20	Reportes para el módulo proveedores	12
SPRINT 6		**60**
HU-21	Gestionar clientes	12
HU-22	Gestionar venta	24
HU-23	Consultar ventas por día, fecha, mes	12
HU-24	Reportes para el módulo ventas	12
SPRINT 7		**60**
HU-25	Administrar caja	24
HU-26	Historial de caja	12
HU-27	Generar Inventario	24
SPRINT 8		**60**
HU-28	Resumen de saldos y movimiento de productos	30
HU-29	Graficas estadísticas de compras y ventas	30
SPRINT 9		**60**
HT-07	Capacitación de usuarios	18

ID	Descripción	Esfuerzo
HT-08	Documentación final del Sistema	24

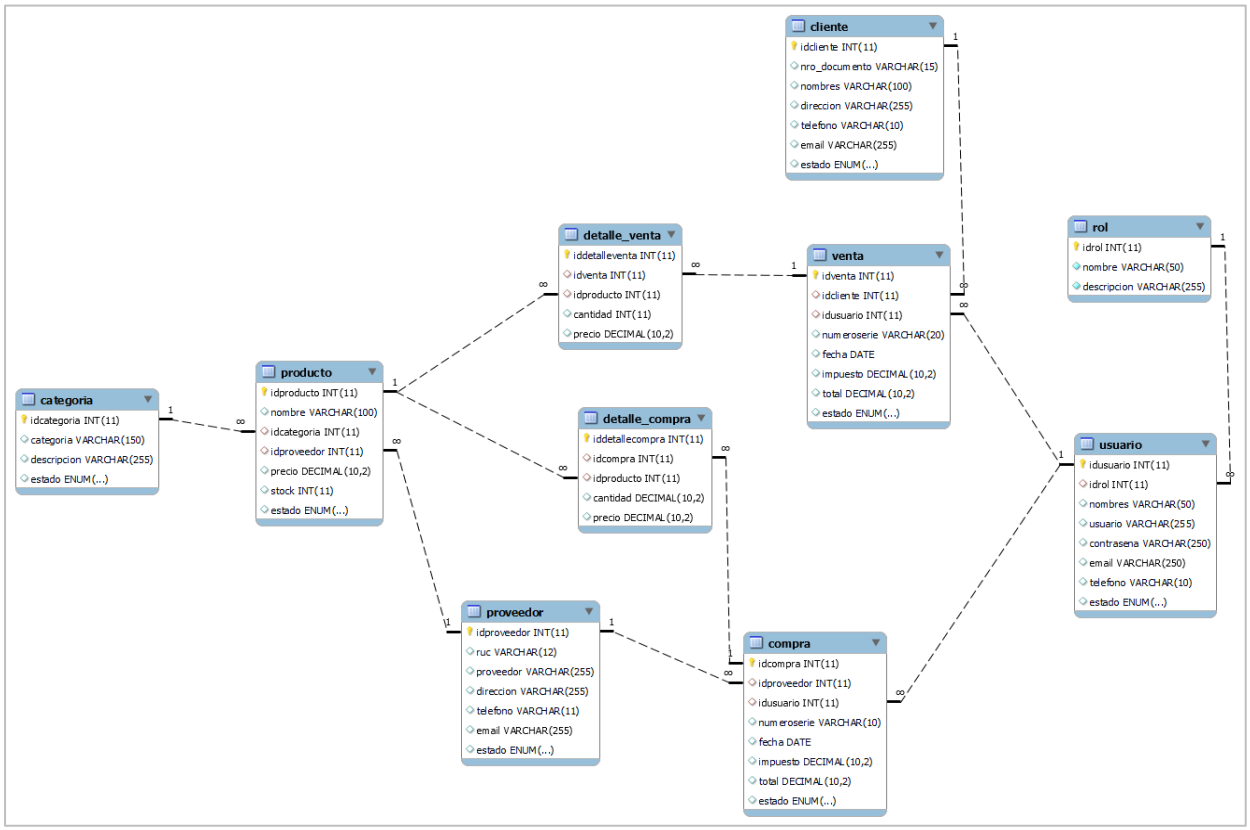
Finalizada la especificación de requisitos se procedio a la codificación mediante el patron MVC, a continuación, se detallan los hitos más importantes:

Tabla 4. Estándar de codificación.

Componente	Nombre	Descripción
Clases	categoriaController	Nombre de la clase y luego nombre del módulo al que pertenece.
Variables	String categoría	Tipo de variable y luego asignar un nombre al atributo
Métodos	categoriaBuscar ()	Nombre de la clase seguido del nombre de método.

La base de datos se diseñó conforme a los requisitos e historias de usuarios. En la siguiente figura, se presenta el modelo físico, donde se pueden identificar las entidades, atributos y sus relaciones.

Figura 1. Diagrama de bases de datos



Finalizada la especificación de las historias de usuario y los diseños del software, se procedió a la codificacióny a expresar estas funcionalidades en códigos de prueba. El Framework JUnit proporciona la API necesaria para llevar a cabo pruebas unitarias de cada requisito. Cada prueba permite evaluar si

un método dentro de un módulo funciona correctamente y verificar el comportamiento de los métodos de dicho módulo.

Para realizar las pruebas unitarias a las funciones del sistema, es necesario crear la clase de la entidad a analizar. Siguiendo el proceso de implementación del marco de trabajo TDD, la prueba que se implementa en las fuentes del proyecto debe fallar como primer paso. Luego se realiza la refactorización y se corrigen los inconvenientes presentados.

A continuación, se presenta como ejemplo la prueba "Agregar productos", seguida de un detalle de cada uno de los pasos hasta obtener la ejecución satisfactoria. El proceso para ejecutar una prueba con la técnica TDD es el siguiente.

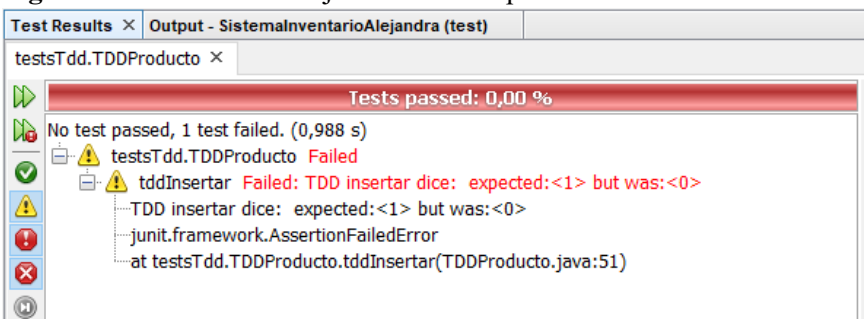
Figura 2. Prueba unitaria para el método "agregar productos"

```
@Before
public void init() {
    pdao = new ProductoDAO();
    objProducto = new Producto();
    cdao = new CategoriaDAO();
    objCategoria = new Categoria();
    provdao = new ProveedorDAO();
    objProveedor = new Proveedor();
}

/*
insertar un nuevo producto
 */
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    objProducto.setNombre("test nombre insertar");
    objCategoria = cdao.buscarCategoria(5);
    objProducto.setIdcategoria(objCategoria);
    objProveedor = provdao.buscarProveedor(7);
    objProducto.setIdproveedor(objProveedor);
    objProducto.setPrecio(10);
    objProducto.setStock(10);
    result = pdao.agregar(objProducto);
    Assert.assertEquals(1, result);
}
```

Nótese que dentro de la prueba unitaria se incluye la etiqueta @Test y el método assertEquals los mismos que indican que se está aplicando un test unitario. La ejecución de las pruebas consiste en verificar si pasan o no, cabe mencionar que no siempre las pruebas se ejecutarán satisfactoriamente, ya que el objetivo de TDD es hacerlas fallar.

Figura 3. Resultado de la ejecución de las pruebas unitarias



Para corregir el error encontrado durante la ejecución del test, es necesario localizar la clase Java y la línea específica donde está mal codificada. En este caso, el problema se soluciona agregando un bloque de excepciones try y catch dentro del método agregarProducto de la clase ProductoDAO.

Figura 4. Refactorización del método Agregar en la clase ProductoDAO

```
public int agregar(Producto producto) {
    int r = 0;
    String sql = "INSERT INTO producto(nombre,idcategoria,idproveedor,precio,stock) values (?,?,?,?,?)";
    try {
        con = conectar.getConnection();
        ps = con.prepareStatement(sql);
        ps.setString(1, producto.getNombre());
        ps.setInt(2, producto.getIdcategoria().getIdcategoria());
        ps.setInt(3, producto.getIdproveedor().getIdproveedor());
        ps.setDouble(4, producto.getPrecio());
        ps.setInt(5, producto.getStock());
        r = ps.executeUpdate();
        con.close();
        if (r == 1) {
            return 1;
        } else {
            return 0;
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return r;
}
```

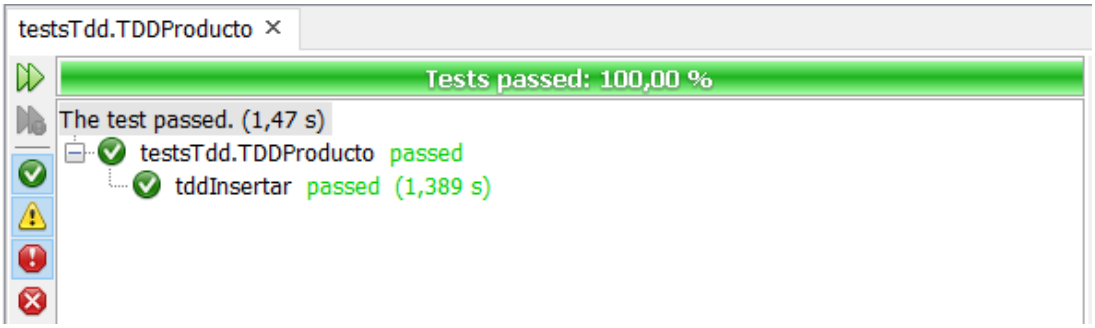
De forma similar se modifica el código de la prueba unitaria, agregando una excepción y las cláusulas de try y cacth.

Figura 5. Refactorización de la prueba unitaria

```
@Test
public void tddInsertar() throws Exception {
    int result = 0;
    try {
        objProducto.setNombre("test nombre insertar");
        objCategoria =  cdao.buscarCategoria(5);
        objProducto.setIdcategoria(objCategoria);
        objProveedor =  provdao.buscarProveedor(7);
        objProducto.setIdproveedor(objProveedor);
        objProducto.setPrecio(10);
        objProducto.setStock(10);
        result = pdao.agregar(objProducto);
    } catch (Exception e) {
        System.out.println("public void tddInsertar() dice: " + e.getMessage
    }
    Assert.assertEquals("TDD insertar dice: ", 1, result);
}
```

Finalmente, el código se ejecuta correctamente.

Figura 6. Ejecución de las pruebas unitarias

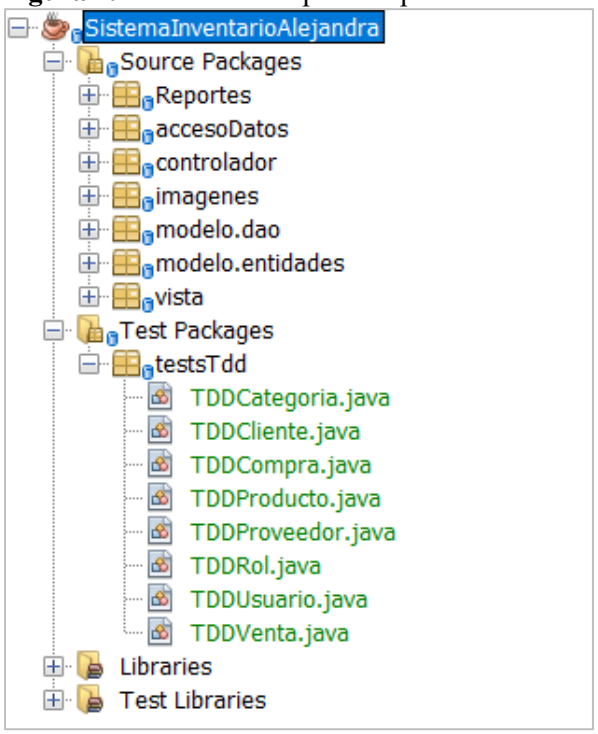


Al concluir la implementación de la metodología de desarrollo de software orientada por pruebas, es fundamental destacar que este documento presenta únicamente algunos ejemplos de pruebas unitarias. Estos ejemplos se proporcionan con el objetivo de ilustrar el proceso que sigue el marco de trabajo TDD.

RESULTADOS Y DISCUSIÓN

Como se mencionó en la fundamentación teórica, "Test Driven Development" es una técnica de programación extrema en la que se escriben las pruebas unitarias antes de cualquier línea de código funcional. Se creó una clase de pruebas para cada uno de los puntos funcionales del sistema, en la que se implementaron las pruebas unitarias necesarias. La siguiente figura ilustra este paralelismo.

Figura 7. Clases creadas para las pruebas TDD



Cada clase de prueba incluye una sección para la declaración de variables globales utilizadas en toda la clase. Además, contiene una operación definida en su correspondiente clase DAO, es decir, la implementación de cada uno de los métodos que contienen las pruebas unitarias para el código funcional.

Es importante destacar que, para determinar si un método de prueba es exitoso o no, se utilizan los métodos assert para comparar los valores esperados con los valores reales, o para provocar deliberadamente que el método de prueba falle.

La anotación @Before indica que el método adjunto se ejecutará antes de cualquier prueba en la clase. Se emplea principalmente para configurar los objetos necesarios para garantizar la correcta ejecución de las pruebas.

Figura 8. Declaración del método init con la anotación @Before

```
/*
inicilaizar los objetos
 */
@Before
public void init() {
    pdao = new ProductoDAO();
    objProducto = new Producto();
    cdao = new CategoriaDAO();
    objCategoria = new Categoria();
    provdao = new ProveedorDAO();
    objProveedor = new Proveedor();
}
```

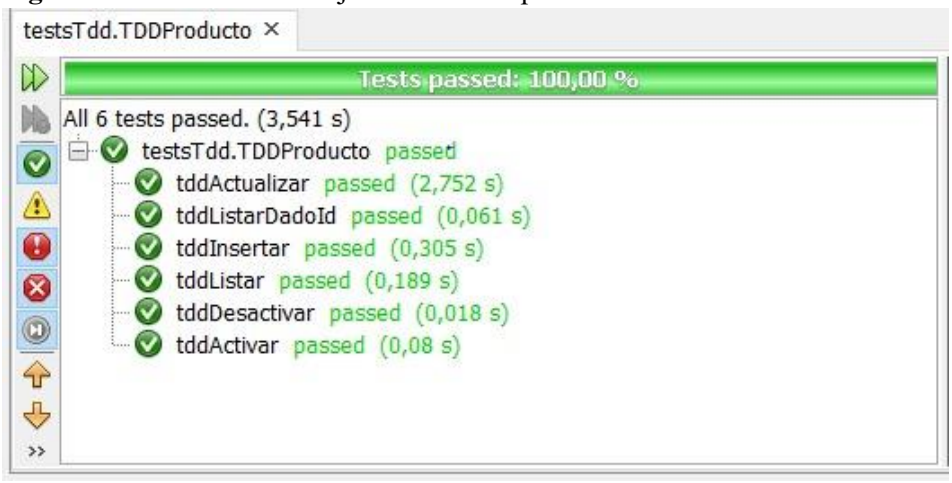
@Test indica que el método adjunto es una prueba unitaria, lo que le permite utilizar cualquier nombre de método para llevar a cabo la prueba.

Figura 9. Definición de la prueba “listar todos los productos”

```
/*
Listar todos los productos
*/
@Test
public void tddListar() {
    List<Producto> lstproductos = new ArrayList<>();
    try {
        lstproductos = dao.listar();
    } catch (Exception e) {
        System.out.println("Tdd listar dice: " + e.getMessage());
    }
    Assert.assertNotNull("Tdd listar productos", lstproductos);
}
```

El resultado obtenido por las pruebas tanto exitosas como erróneas se verá de forma similar como la siguiente figura.

Figura 10. Resultado de la ejecución de las pruebas unitarias



Después de estudiar TDD junto con el framework JUnit, se observa que esta técnica no solo abarca la prueba de la aplicación, sino que también guía el diseño de la misma, mejorando así el paradigma de desarrollo del software y obteniendo un código de calidad. El framework JUnit facilita la adaptabilidad de los sistemas al cambio al permitir la detección y el aislamiento fácil de las pruebas unitarias realizadas en cada parte del sistema.

El uso de TDD se complica en situaciones que requieren pruebas completas de funcionalidad del sistema, como el uso de interfaces gráficas de usuario (GUI), programas que trabajan con bases de datos relacionales y aquellos que operan en diferentes configuraciones de red. La aplicación de TDD implica un cambio de paradigma en el desarrollo de software, ya que requiere la escritura de pruebas unitarias antes del código fuente. Esto puede resultar desafiante para los desarrolladores con poca experiencia, pero no se descarta la posibilidad de que, con más práctica, puedan utilizar esta técnica en el futuro tanto en el ámbito académico como en el laboral.

TDD es útil en las pruebas de regresión, ya que los desarrolladores pueden utilizarlo para determinar qué cambios afectaron las funcionalidades de la aplicación. A diferencia de las pruebas unitarias, TDD no realiza pruebas de manera aislada, sino que cubre funcionalidades consecutivas y permite la reutilización del código en el futuro.

CONCLUSIONES

A través de un análisis exhaustivo de la documentación disponible sobre la Metodología de Desarrollo de Software Orientado por Pruebas (TDD), se determina que esta técnica se centra en el diseño e implementación de pruebas unitarias. Estas pruebas generalmente se basan en la especificación de los requerimientos funcionales del software o, en el caso de la Metodología Scrum, en las historias de usuario. Se observa que TDD puede integrarse con cualquier framework, y en el caso específico de JUnit, su utilización no se limita únicamente a la realización de pruebas de la aplicación, sino que también sirve como base para el diseño de la misma, asegurando la correcta lógica del código y su capacidad para superar las pruebas.

La ejecución de las pruebas unitarias como parte de la implementación de la metodología TDD permitió clarificar la lógica de negocio del software y proporcionó una visión preliminar del producto final. Al finalizar la implementación de TDD, estas pruebas unitarias se convirtieron en funcionalidades del software, lo que permitió al equipo de desarrollo probar diferentes partes del software de manera independiente, acelerando así el proceso de desarrollo.

El sistema de control de inventario para el Almacén Alejandra se desarrolló e implementó utilizando el código refactorizado, que fue el resultado de la implementación de las pruebas unitarias definidas a través de la Metodología TDD. La combinación de Scrum y TDD se integró de manera coherente en el desarrollo del caso práctico, lo que permitió un control y una distribución óptima de las actividades de trabajo. Esta combinación también permitió al equipo de desarrollo enfocarse en las actividades prioritarias, lo que resultó en una aplicación robusta y fácil de usar para el usuario final. La elección de Java y MySQL como tecnologías complementarias contribuyó a este éxito..

REFERENCIAS BIBLIOGRAFICAS

Adewole, A. (2018). *C# and .NET Core Test Driven Development: Dive into TDD to create flexible, maintainable, and production-ready .NET Core applications.* Packt Publishing.

Amaya, Y. D. (2013). Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. *Revista de Tecnología - Journal Technology*, *12*(2), 111-124.

Beck, K. (2002). *Test Driven Development: By Example.* Addison-Wesley.

Blé, C. (2020). *Diseño Ágil con TDD.* Lean Mind. Obtenido de https://leanpub.com/tdd-en-castellano

Casillas, L. A., Ginestà, M. G., & Pérez, Ó. (2014). Bases de datos en MySQL. *Universitat oberta de Catalunya*.

Castro, D. V. (2016). *Sistema de capacitación virtual para la implementación de la metodología de desarrollo de software test driven devolopmen. [Tesis de Pregrado]*. Universidad Central del Ecuador. Obtenido de http://www.dspace.uce.edu.ec/bitstream/25000/7752/1/T-UCE-0011-298.pdf

Farcic, V., & García, A. (2015). *Test-Driven Java Development.* Packt Publishing.

Gałęzowski, G. (2016). *Test-Driven Development: Extensive Tutorial.* Lean Publishing. Obtenido de https://openlibra.com/es/book/download/test-driven-development-extensive-tutorial

Garrido, P. (2015). *Comenzando a programar con JAVA (Spanish Edition).* Universitas Miguel Hernández.

Grenning , J. (2020). *Test Driven Development for Embedded C: Building Hihg Quality Embedded Software (Pragmatic Programmers).* Pragmatic Bookshelf.

Grenning, J. W. (2011). *Test Driven Development for Embedded C.* Pragmatic bookshelf.

Guerra, O. P., & Lema, L. E. (2018). *Implementar un sistema para procesar los datos que se levantan en el inventario de salud con información de las afecciones a la salud, clasificación c10, utilizando una plataforma java, postgresql. [Tesis de pregrado]*. Obtenido de https://dspace.ups.edu.ec/handle/123456789/15335

Haq, M. Z. (2017). *Angular Test-Driven Development* (Vol. 2nd Revised Edition). Packt Publishing.

Laínez, J. R. (2015). *Desarrollo de Software ÁGIL: Extreme Programming y Scrum.* Createspace Independent Publishing Platform.

Lee , G. (16 de Octubre de 2016). *Tipos de pruebas de software: diferencias y ejemplos.* Obtenido de https://www.loadview-testing.com/es/blog/tipos-de-pruebas-de-software-diferencias-y-ejemplos/

Microsoft Developers Network. (2021). *Conceptos básicos de las pruebas unitarias.* Obtenido de https://docs.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2019&redirectedfrom=MSDN&viewFallbackFrom=vs-2015

Orozco Alvarez, E. A. (2018). *Desarrollo del módulo PIM-PSM versión 5.0 de la herramienta jMDA [Tesis de pregrado]*. Universidad Central "Martha Abreu" de Las Villas. Obtenido de https://dspace.uclv.edu.cu/bitstream/handle/123456789/9953/Tesis%20Eduardo%20Orozco.pdf?sequence=1&isAllowed=y

Parra, D. C., & Ramírez, J. M. (2018). *DISEÑO DESARROLLO E IMPLEMENTACION DE SOFTWARE Y APLICATIVO MOVIL PARA LA ADMINISTRACION Y GESTION DE VENTA Y PREVENTA DE LA DISTRIBUIDORA BUITRAGO.* Universidad Piloto de Colombia - Ingeniería de Sistemas.

Pozo, T., Aucancela, C., Hinojosa, C., & Abdelrahaman, A. (2011). Sistema Web de Asignación de Aulas de los Laboratorios de Computación de la ESPE, Aplicando la Metodología Agile Unified Process (AUP), utilizando el Framework Junit. *Revista DECC Report, Tendencias en Computación*, *3*(1), 6-14. Obtenido de https://journal.espe.edu.ec/ojs/index.php/geeks/article/download/252/229

Ress, A. P., de Oliveira, R., & Salerno, M. S. (2013). Test-Driven Development as an Innovation Value Chain. *Journal of Technology Management & Innovation*, *8*(1). Obtenido de http://dx.doi.org/10.4067/S0718-27242013000300010

Salazar, J. C., Tovar, Á., Linares, J. C., Lozano, A., & Valbuena, Y. L. (2018). Scrum versus XP: similitudes. *TIA*, *6*(2), 29-37. Obtenido de https://revistas.udistrital.edu.co/index.php/tia/article/view/10496

Storani, M. (20 de Junio de 2008). *TDD – Test Driven Development*. Obtenido de https://mauriziostorani.wordpress.com/2008/06/20/tdd-test-driven-development/

Sznajdleder, P. (2013). *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones* (Segunda ed.). Buenos Aires: Alfaomega Grupo Editor Argentino. Obtenido de https://www.academia.edu/14584688/Java_a_fondo_Sznajdleder_Pablo

Torres-Corredor, O. I. (2017). *Aplicación y evaluación de la metodología desarrollo orientado por pruebas (TDD), caso de estudio: spot.co. [Tesis de Maestría]*. Universidad Internacional de la Rioja. Obtenido de https://reunir.unir.net/handle/123456789/6562

Vaca et al. (2014). Test-Driven Development - Una aproximación para entender su utilidad en el proceso de desarrollo de Software. *WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación*, 570 - 574. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/41604/Documento_completo.pdf?sequence=1

Veintimilla, A. J., & Cuenca, L. F. (2014). *Estudio de la técnica Test Driven Development (TDD) y desarrollo del sistema para la administración de consultorios médicos. [Tesis de pregrado]*. Universidad de las Fuerzas Armadas ESPE. . Obtenido de https://repositorio.espe.edu.ec/bitstream/21000/9094/1/T-ESPE-048066.pdf