Desarrollo y Evaluación de un Sistema de Punto de Venta para Pequeñas y Medianas Empresas Utilizando Arquitectura en Capas

A. LINDAO[1], A. LOPEZ[1], J. MEDINA[1], J. QUINDE[1], J. VILLACRES[1], J. ZAMBRANO[1]
[1]Facultad de Ciencias Matemáticas y Físicas, Universidad de Guayaquil, Guayaquil, Ecuador

Emails: alan.lindaoh@ug.edu.ec, anthony.lopezr@ug.edu.ec, julio.medinac@ug.edu.ec, jeremy.quindegal@ug.edu.ec, joshua.villacresbot@ug.edu.ec, juan.zambranoqui@ug.edu.ec

RESUMEN En el presente artículo, se describe el desarrollo de un sistema de punto de venta (POS) diseñado para pequeñas y medianas empresas, abordando los desafíos y soluciones implementadas durante el proceso. Utilizando una arquitectura en capas con Estratificación Estricta, se dividió el sistema en tres capas principales: la capa de datos, la capa de negocio y la capa de presentación. Cada capa fue diseñada para cumplir con responsabilidades específicas, mejorando la modularidad y mantenibilidad del sistema. La capa de datos empleó el patrón Repositorio, la capa de negocio utilizó el patrón EntityState, y la capa de presentación adoptó la arquitectura Modelo-Vista-Modelo de Vista (MVVM). Para el desarrollo, se utilizaron tecnologías como C# 6.0, SQL Server 2022, y Visual Studio, entre otras. El proyecto enfrentó desafíos relacionados con la integración entre capas, el manejo del estado de entidades y el rendimiento de consultas, los cuales fueron superados mediante soluciones efectivas y pruebas exhaustivas. Se realizaron pruebas unitarias, de integración para asegurar la calidad del sistema. El proceso proporcionó valiosas lecciones sobre la importancia de la planificación, el modelado previo, la flexibilidad y el manejo de errores. Los resultados y la experiencia obtenida enriquecieron nuestras habilidades en desarrollo de software.

PALABRAS CLAVES Arquitectura en Capas, C# 6.0, Desarrollo de Software, EntityState, Estratificación Estricta, Modelo-Vista-Modelo de Vista (MVVM), Optimización de Consultas, Patrón Repositorio, Pruebas de Integración, Pruebas Unitarias, SQL Server 2022, Visual Studio

I. INTRODUCCIÓN

En la actualidad, las empresas independientemente de su tamaño enfrentan el desafío de gestionar eficientemente la gran cantidad de información generada por sus usuarios. La solución más común es implementar sistemas personalizados que les permitan manejar sus operaciones de manera efectiva.

Como futuros ingenieros de software, tenemos la responsabilidad de comprender las necesidades de nuestros clientes y desarrollar sistemas que las satisfagan. En el marco de la asignatura de Construcción de Software, decidimos abordar este desafío desarrollando un sistema de punto de venta (POS) destinado a pequeñas y medianas empresas.

Nuestro objetivo principal es crear una solución sencilla y eficiente para la gestión de ventas, inventario y clientes. Este proyecto no solo busca proporcionar una herramienta útil para las empresas, sino también ofrecernos la oportunidad de adquirir experiencia práctica en el desarrollo colaborativo de software.

II. ESTIMACIÓN DE COSTOS USANDO EL MODELO COCOMO I

Para realizar una estimación de costos utilizando el modelo COCOMO básico, consideramos las siguientes características de nuestro proyecto:

- **Tamaño del Proyecto:** 8,000 líneas de código (8 KLOC)
- **Tipo de Proyecto:** Orgánico

A. CÁLCULO DE ESFUERZO (PERSONA-MESES)

La fórmula para calcular el esfuerzo en un proyecto Orgánico es:

$$Esfuerzo(PM) = 2.4 \times (KLOC)^{1.05}$$

Aplicamos los datos:

$$Esfuerzo = 2.4 \times (8)^{1.05}$$

Realicemos el cálculo:

$$(8)1.05 \approx 8.39$$

$$Esfuerzo \approx 2.4 \times 8.39 \approx 20.14\ persona - meses$$

B. CÁLCULO DE TIEMPO DE DESARROLLO (MESES)

La fórmula para calcular el tiempo de desarrollo es:
$Tiempo\ de\ Desarrollo(TDEV) = 2.5 \times (Esfuerzo)^{0.38}$

Aplicamos el esfuerzo calculado:
$Tiempo\ de\ Desarrollo = 2.5 \times (20.14)^{0.38}$

Realicemos el cálculo:
$(20.14)^{0.38} \approx 3.34$

$Tiempo\ de\ Desarrollo \approx 2.5 \times 3.34 \approx 8.35\ meses$

C. NÚMERO DE PERSONAS REQUERIDAS

La fórmula para calcular el número de personas requeridas es:
$N\acute{u}mero\ de\ Personas = \frac{Esfuerzo}{Tiempo\ de\ Desarrollo}$

Aplicamos los valores calculados:
$N\acute{u}mero\ de\ Personas = \frac{20.14}{8.35} \approx 2.41\ personas$

D. CÁLCULO DE COSTOS

Para calcular los costos, consideramos una tarifa promedio de $500 por persona-mes.

$Costo\ Total = Esfuerzo \times Tarifa\ por\ persona - mes$
$Costo\ Total = 20.14 \times 500 = \$10{,}070$

E. RESUMEN DE LA ESTIMACIÓN

- **Esfuerzo Estimado**: 20.14 persona-meses
- **Tiempo de Desarrollo Estimado:** 8.35 meses
- **Número de Personas Requeridas Estimado:** 2.41 personas
- **Costo Total Estimado:** $10,070

III. PLANIFICACIÓN Y DISEÑO

A. RECOPILACIÓN DE REQUISITOS

Para identificar los requisitos funcionales y no funcionales del sistema, adoptamos un enfoque sistemático que incluyó técnicas de recopilación de datos. Inicialmente, empleamos la técnica del brainstorming o lluvia de ideas, que nos permitió capturar una amplia gama de opiniones y sugerencias de todos los integrantes del equipo. Esta técnica facilitó una discusión abierta y creativa sobre las posibles funcionalidades y características del sistema.

Una vez recopiladas las ideas, organizamos y categorizamos los requisitos en dos grandes grupos: funcionales y no funcionales. Los requisitos funcionales se centraron en las capacidades y características específicas que el sistema debía ofrecer, como el procesamiento de ventas y la gestión de usuarios. Los requisitos no funcionales, por otro lado, abordaron aspectos como el rendimiento, la usabilidad, la seguridad y la escalabilidad del sistema.

Para garantizar que solo las funcionalidades más relevantes y cruciales fueran seleccionadas, descartamos aquellas funcionalidades que no fueran una prioridad obteniendo como resultado de este proceso una lista consolidada de requisitos que sirvió como base para la fase de diseño y desarrollo del sistema.

B. DISEÑO DEL SISTEMA

Diseñamos una arquitectura en capas utilizando Estratificación Estricta, un enfoque que promueve la separación clara de responsabilidades entre diferentes niveles del sistema. Esta arquitectura nos permitió dividir la aplicación en tres capas principales:

1. **Capa de Datos**: Implementamos el patrón Repositorio en esta capa para gestionar el acceso y la manipulación de los datos.
2. **Capa de Negocio**: En esta capa, utilizamos el patrón EntityState para manejar el estado de las entidades a lo largo de las operaciones de negocio.
3. **Capa de Presentación**: Adoptamos la arquitectura Modelo-Vista-Modelo de Vista (MVVM) para la capa de presentación.

IV. HERRAMIENTAS Y TECNOLOGÍA

A. LENGUAJES DE PROGRAMACIÓN

Para el desarrollo de nuestro sistema, seleccionamos C# 6.0 como el lenguaje de programación principal. La elección de C# se basó en su robustez y amplia adopción en el desarrollo de aplicaciones empresariales.

B. BASE DE DATOS

Utilizamos SQL Server 2022 para la gestión de la base de datos. Esta versión nos proporcionó características avanzadas de seguridad y rendimiento.

C. ENTORNO DE DESARROLLO

Para el desarrollo del software, empleamos Visual Studio como nuestro entorno de desarrollo integrado (IDE), el cual nos proporcionó una plataforma robusta y rica en características para escribir, depurar y probar nuestro código. Además, utilizamos GitHub como nuestro repositorio de código, este nos facilitó la colaboración entre los miembros del equipo.

C. PAQUETES Y LIBRERÍAS

Para desarrollar nuestro proyecto hicimos uso de varios paquetes y librerías, como SqlClient, proveedor de datos de .NET para SQL Server, que nos permitió conectar nuestra aplicación con la base de datos y realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) de manera eficiente.
NUnit, el cual usamos como nuestro framework de pruebas unitarias para .NET. NUnit facilitó la creación y ejecución de pruebas automatizadas para asegurar la calidad del código.
NUnit3TestAdapter, el cual se integró con la ventana de pruebas unitarias de Visual Studio, permitiéndonos ejecutar y visualizar los resultados de nuestras pruebas unitarias directamente en el IDE.

Microsoft.NET.Test.Sdk proporcionó la infraestructura necesaria para ejecutar pruebas unitarias en proyectos .NET dentro de Visual Studio.

V. DESARROLLO

A. DIVISIÓN DEL TRABAJO

Nos organizamos en tres equipos: capa de presentación, capa de negocio y capa de datos. Cada equipo tenía sus tareas específicas, pero manteníamos reuniones semanales para coordinar el progreso y resolver problemas con los que nos íbamos topando.

B. DESARROLLO DE LA CAPA DE DATOS

Implementamos el patrón Repositorio para gestionar el acceso a la base de datos. Este patrón proporcionó una interfaz abstracta para interactuar con los datos, permitiendo operaciones CRUD (Crear, Leer, Actualizar y Eliminar) de manera eficiente. La capa de datos se encargó de las consultas a la base de datos, la ejecución de comandos y la manipulación de datos, garantizando que la lógica de acceso a datos estuviera desacoplada de la lógica de negocio.

C. DESARROLLO DE LA CAPA DE NEGOCIO

En esta capa, utilizamos el patrón EntityState para gestionar el estado de las entidades a lo largo de las operaciones del sistema. EntityState permitió el seguimiento y la sincronización de los cambios en las entidades, asegurando que las actualizaciones se reflejasen correctamente en la base de datos y facilitando la implementación de las reglas de negocio.

D. DESARROLLO DE LA CAPA DE PRESENTACIÓN

Aplicamos la arquitectura Modelo-Vista-Modelo de Vista (MVVM) para separar la lógica de la interfaz de usuario de la lógica de presentación. La capa de presentación se encargó de la interacción con el usuario, presentando la información de manera clara y gestionando las solicitudes de entrada. MVVM permitió una gestión más limpia y modular de la interfaz de usuario, facilitando el desarrollo y mantenimiento de las vistas y sus correspondientes modelos de vista.

D. DESARROLLO DE FUNCIONALIDADES

Durante el desarrollo, nos centramos en implementar las funcionalidades clave del sistema, asegurando que cada componente funcionara de acuerdo con los requisitos definidos:

1. **Gestión de Productos**: Implementamos funcionalidades para agregar, editar, eliminar y visualizar productos. Esto incluyó la creación de interfaces para la entrada de datos y la integración con la capa de datos para almacenar y recuperar información.
2. **Gestión de Usuarios y Clientes**: Desarrollamos módulos para manejar la información de usuarios y clientes, incluyendo la autenticación de usuarios y la gestión de perfiles de cliente.
3. **Procesos de Venta**: Implementamos las funcionalidades necesarias para registrar y gestionar ventas.
4. **Pruebas y Validación**: Realizamos pruebas unitarias utilizando NUnit para verificar la funcionalidad de cada componente del sistema. Las pruebas se integraron en el flujo de trabajo de desarrollo para asegurar que cada nuevo cambio no introdujera errores en el sistema.

VI. DESAFÍOS Y SOLUCIONES

Durante el desarrollo del sistema, enfrentamos varios desafíos que requirieron soluciones innovadoras para garantizar el éxito del proyecto. A continuación, se describen algunos de los principales problemas y cómo los abordamos.

A. INTEGRACIÓN ENTRE CAPAS

El desafío era asegurar que las diferentes capas del sistema interactuaran correctamente sin violar los principios de separación de responsabilidades, para lo cual implementamos interfaces bien definidas entre la capa de datos, la capa de negocio y la capa de presentación. Ajustamos las interfaces según fuera necesario para resolver problemas de integración.

B. MANEJO DEL ESTADO DE ENTIDADES

Debíamos sincronizar correctamente el estado de las entidades entre la capa de negocio y la base de datos, especialmente con cambios concurrentes. Adoptamos el patrón EntityState para gestionar el seguimiento y actualización del estado de las entidades. Implementamos mecanismos para manejar conflictos de concurrencia y aseguramos la coherencia de los datos mediante transacciones adecuadas.

C. VALIDACIÓN Y MANEJO DE ERRORES

Con esto buscábamos garantizar que el sistema manejara errores de manera robusta y proporcionara mensajes de validación claros al usuario. Implementamos un manejo de errores centralizado y validaciones en la capa de presentación. Desarrollamos pruebas exhaustivas para validar los casos de borde y asegurar que los errores fueran gestionados de manera adecuada.

VII. PRUEBAS Y EVALUACIÓN

Para asegurar la calidad y el correcto funcionamiento del sistema, llevamos a cabo un proceso exhaustivo de pruebas y evaluación:

A. PRUEBAS UNITARIAS

Utilizamos NUnit para realizar pruebas unitarias en los componentes individuales del sistema. Estas pruebas verificaron la correcta implementación de funciones y métodos en cada capa. Las pruebas unitarias nos permitieron

identificar y corregir errores, asegurando que cada componente funcionara según lo esperado.

B. PRUEBAS DE INTEGRACIÓN

Realizamos pruebas de integración para validar la interacción entre las distintas capas del sistema. Estas pruebas confirmaron que los datos se transmitieran correctamente entre la capa de datos y la base de datos.

Las pruebas de integración ayudaron a detectar problemas de comunicación entre capas y permitieron ajustar las interfaces y los mecanismos de integración.

VIII. LECCIONES APRENDIDAS

A lo largo del desarrollo del sistema, aprendimos varias lecciones valiosas:

A. IMPORTANCIA DE LA PLANIFICACIÓN

La planificación detallada y la definición clara de requisitos son cruciales para el éxito del proyecto. Un diseño bien planificado facilita la implementación y reduce el riesgo de problemas durante el desarrollo.

B. BENEFICIOS DEL MODELADO PREVIO

La creación de diagramas de flujo y diagramas ERD antes de comenzar la codificación ayudó a visualizar y resolver problemas de diseño temprano. El modelado previo es una práctica recomendada para evitar problemas en etapas posteriores.

C. NECESIDAD DE PRUEBAS EXHAUSTIVAS

Las pruebas exhaustivas, incluyendo pruebas unitarias y de integración, son esenciales para garantizar la calidad y fiabilidad del sistema. La integración de pruebas en el flujo de trabajo de desarrollo ayuda a identificar y corregir errores de manera oportuna.

D. FLEXIBILIDAD Y ADAPTACIÓN

La capacidad de adaptarse a cambios y resolver problemas imprevistos es fundamental. Aprendimos a ajustar nuestro enfoque y a buscar soluciones creativas para superar los desafíos que surgieron durante el desarrollo.

IX. CONCLUSIÓN

El desarrollo del sistema de punto de venta ha sido una experiencia educativa valiosa que nos ha permitido aplicar y profundizar nuestros conocimientos en arquitectura de software, desarrollo de bases de datos y pruebas de software. La implementación de una arquitectura en capas, junto con el uso de herramientas y tecnologías avanzadas, ha resultado en un sistema robusto y eficiente. Las lecciones aprendidas y los desafíos superados durante el proyecto han enriquecido nuestra experiencia y nos han preparado mejor para futuros proyectos en el campo del desarrollo de software.

X. AGRADECIMIENTOS

Queremos expresar nuestro agradecimiento a nuestro profesor por sus valiosas enseñanzas que fueron fundamentales para afrontar con éxito el desarrollo del proyecto.

Agradecemos también a nuestros compañeros de equipo por su dedicación, esfuerzo y colaboración durante todo el proceso. Su compromiso y trabajo en equipo fueron clave para la realización del sistema, y el ambiente de cooperación y apoyo mutuo contribuyó significativamente al éxito del proyecto.

XI. REFERENCIAS

Aquí incluimos enlaces a la documentación de las herramientas y tecnologías utilizadas en el proyecto:

- **C# 6.0**: Documentación oficial de C#
- **SQL Server 2022**: Documentación oficial de SQL Server
- **Visual Studio**: Documentación oficial de Visual Studio
- **GitHub**: Documentación oficial de GitHub
- **SqlClient**: Documentación de SqlClient
- **NUnit**: Documentación oficial de NUnit
- **NUnit3TestAdapter**: Documentación de NUnit3TestAdapter
- **Microsoft.NET.Test.Sdk**: Documentación de Microsoft.NET.Test.Sdk