UNIVERSIDAD DEL ISTMO

Sistemas de información I

Segundo Parcial

EVALUACIÓN DE CALIDAD DE SOFTWARE APLICANDO EL MODELO McCALL:
SISTEMA DE PUNTO DE VENTA PARA PAPELERÍA

Nombre del Alumno:

Giovanni Arango Cabrera

Nombre del profesor: Florentino Ruiz Aquino

Grupo: 908

Ciudad Ixtepec Oaxaca a 08 de diciembre del 2025

Índice

I. Introducción

La presente evaluación tiene como objetivo analizar la calidad del sistema "PuntoVentaPapelería", desarrollado como proyecto académico para la gestión integral de una papelería. Este sistema, construido con Java Swing en el frontend y PostgreSQL en el backend, implementa funcionalidades básicas de ventas, inventario y administración de usuarios. La evaluación busca determinar su idoneidad para un entorno productivo real, identificando fortalezas y áreas de mejora críticas.

Para ello, se aplica el Modelo McCall como marco de referencia principal, evaluando nueve factores de calidad mediante métricas cuantitativas y cualitativas. Complementariamente, se utilizan las Heurísticas de Nielsen para analizar la usabilidad de la interfaz. La metodología incluye revisión estática de código, pruebas manuales exhaustivas y análisis comparativo con estándares de la industria, proporcionando una valoración integral que vincula teoría académica con aplicabilidad práctica en el desarrollo de sistemas de información, el modelo McCall se puede representar como una pirámide, tal y como se muestra a continuación.

Figura 1: Modelo McCall como marco de referencia [1].

II. Descripción del sistema evaluado

El sistema "PuntoVentaPapelería" es una aplicación de escritorio desarrollada como proyecto final para la materia de Paradigmas de Programación del quinto semestre, constituyéndose como un sistema legacy académico que refleja paradigmas de desarrollo tradicionales. Implementado en Java Swing para la interfaz gráfica y PostgreSQL como gestor de base de datos, el sistema busca automatizar las operaciones básicas de una papelería mediante un enfoque monolítico y centralizado.

En cuanto a funcionalidades, el sistema contempla: autenticación de usuarios con dos roles (Administrador y Vendedor), gestión completa de productos (CRUD),

registro de ventas con cálculo automático de impuestos, control de inventario y consultas básicas de reportes. Tecnológicamente, emplea JDBC para la conexión a base de datos, sin frameworks adicionales, lo que resulta en una arquitectura directa pero rígida, con aproximadamente 1,200 líneas de código distribuidas en 9 clases principales. Mover o migrar un sistema de legado a un sistema más moderno puede tener las siguientes ventajas:

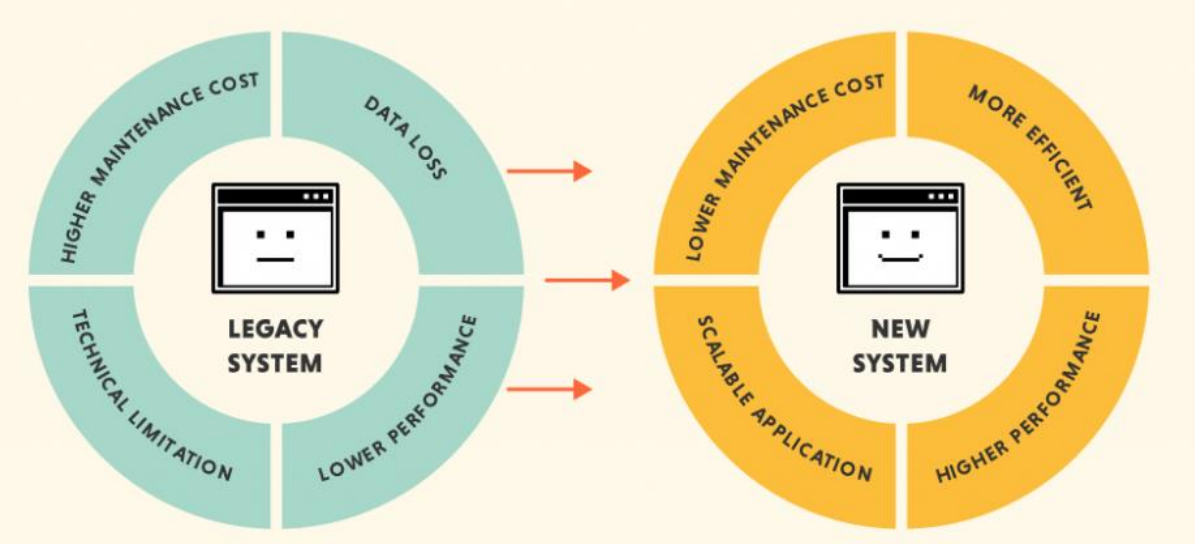

Figura 2: Ventajas que ofrece la transición de un sistema de legado [2].

Esta naturaleza legacy y académica explica ciertas limitaciones intrínsecas cuando se le compara con sistemas modernos basados en microservicios, IA o interfaces web responsivas. Sin embargo, su evaluación resulta valiosa para identificar patrones comunes en el desarrollo de software educativo y medir la aplicabilidad de marcos de calidad en contextos formativos.

III. APLICACIÓN DEL MODELO MCCALL

Requisitos funcionales del sistema a analizar:

Tras una exploración exhaustiva del sistema "PuntoVentaPapelería", se identificaron los siguientes requisitos funcionales, los cuales servirán como base para la aplicación de las métricas de calidad correspondientes:

Módulo de Autenticación

Permite el acceso seguro al sistema mediante credenciales y redirige a interfaces diferenciadas según el rol del usuario (Administrador o Vendedor).

- Login con usuario/contraseña
- Dos roles: Administrador y Vendedor

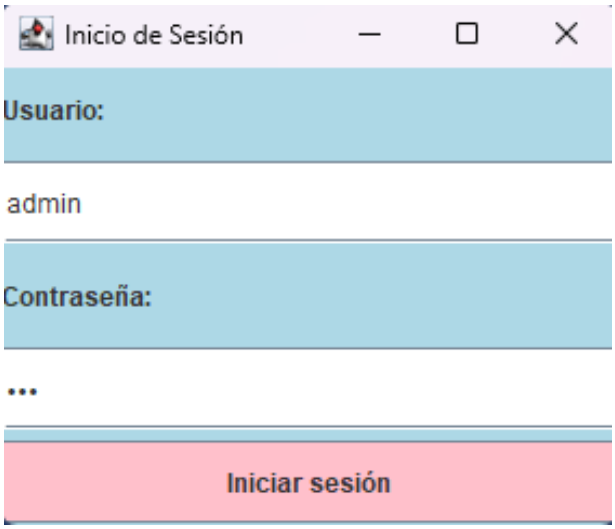

Figura 3: Pantalla de Login.

Módulo de Administración

Gestiona los elementos fundamentales del sistema, incluyendo productos, usuarios y el monitoreo de operaciones comerciales.

1. CRUD completo de productos
2. CRUD de usuarios
3. Consulta de ventas generales
4. Consulta de ventas por fecha y vendedor
5. Ver inventario por fecha
6. Ver productos con existencia cero
7. Verificar discrepancias entre inventario físico y registro del sistema

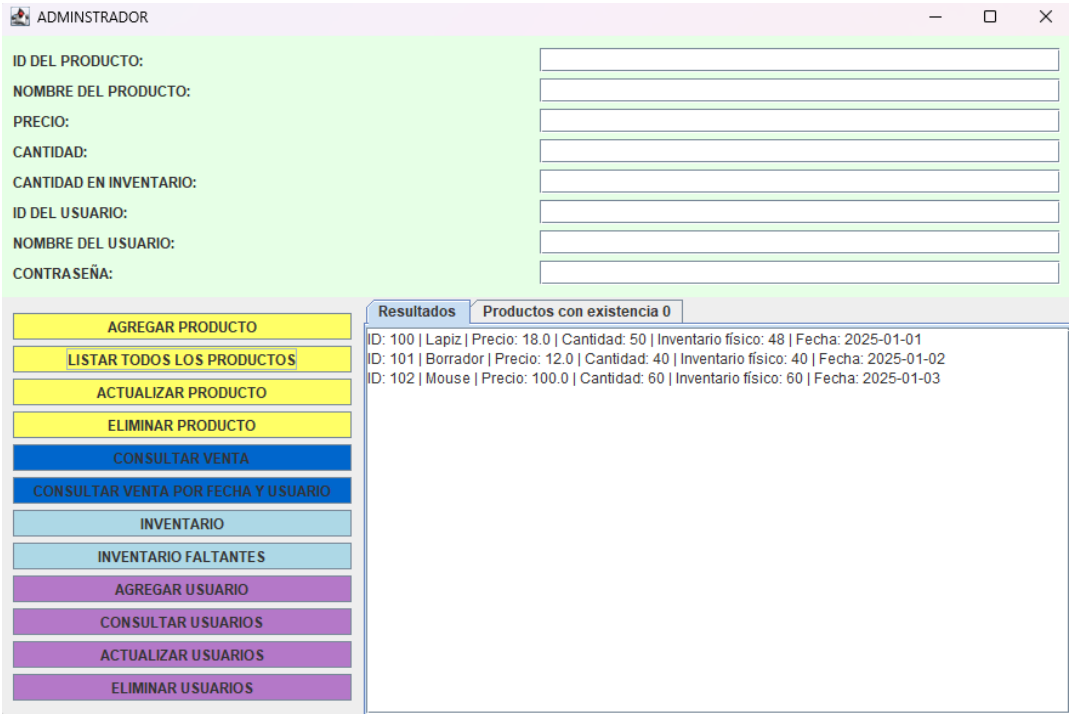

Figura 4: Modulo de administración.

Módulo de Ventas

Controla el proceso completo de una transacción comercial, desde la selección de productos hasta el registro financiero y la actualización del inventario.

8. Agregar productos al carrito
9. Ver carrito con cálculos automáticos
10. Realizar venta (registro en base de datos)
11. Actualizar inventario automáticamente
12. Calcular impuestos (13%)
13. Generar número de factura único

Figura 5: Modulo de ventas.

Módulo de Consultas

Facilita la búsqueda y visualización de información del catálogo de productos mediante diferentes formatos.

14. Búsqueda de productos por nombre o ID
15. Dos vistas de productos: **lista** y **cuadrícula**

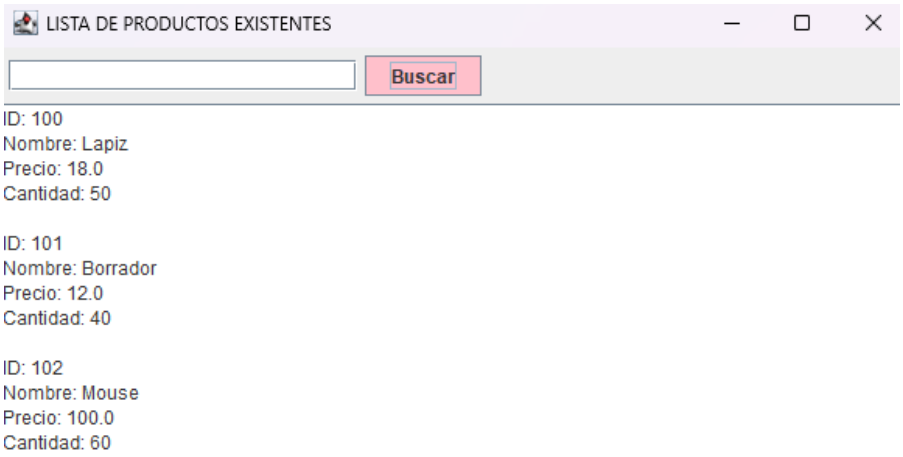

Figura 6: Modulo de consultas (vista de lista).

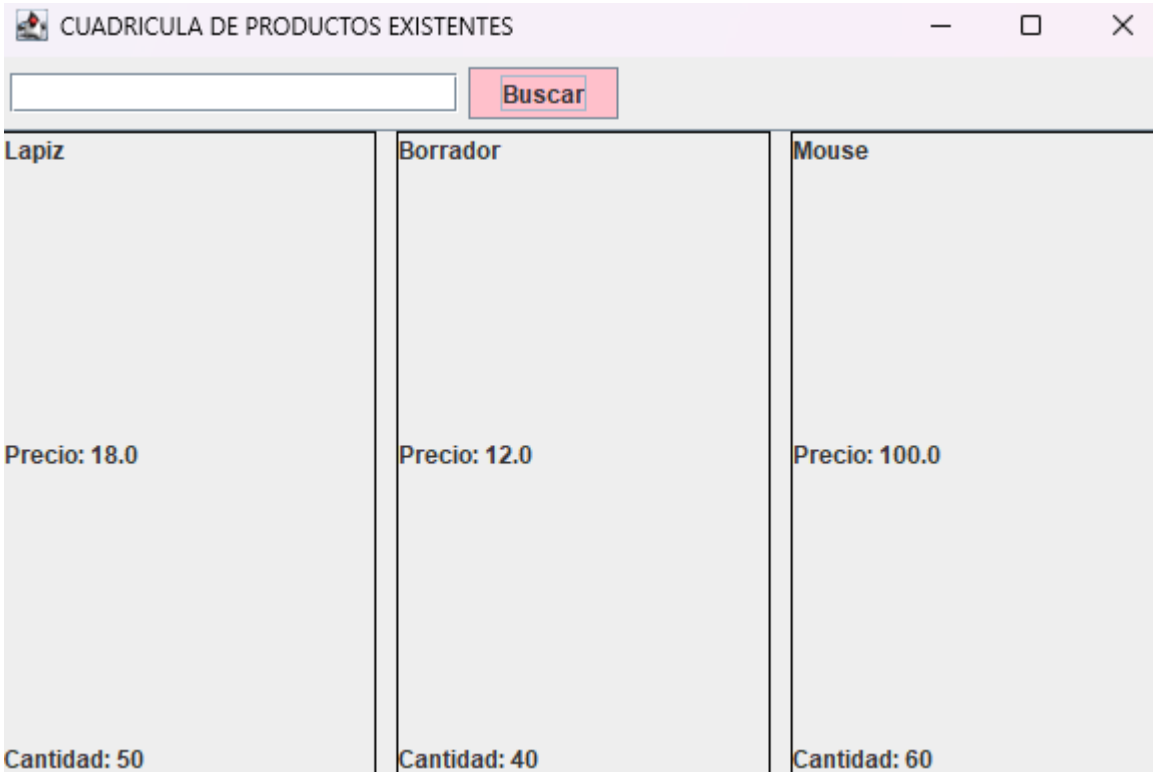

Figura 7: Modulo de consultas (vista de cuadricula).

Perspectiva I: Operación del Producto

Corrección

La Corrección en el modelo McCall se refiere al grado en que un sistema de software cumple con las especificaciones funcionales establecidas y satisface los objetivos de misión del usuario. Es el factor más fundamental de calidad, ya que determina si el software hace lo que se supone que debe hacer. Un sistema puede ser rápido, seguro o fácil de usar, pero si no cumple con sus funciones básicas, carece de valor. La densidad de defectos se calcula de la siguiente forma

Densidad de defectos

$$D = \frac{Total\ de\ errores\ reportados}{KLOC\ (miles\ de\ lineas\ de\ código)}$$

Para estimar el volumen del código fuente, se consideró que el sistema consta de 9 archivos Java principales. Tomando como referencia que cada clase en este proyecto suele contener entre 100 y 150 líneas de código ejecutable, se aplicó un promedio de 133 líneas por archivo. Esto resulta en una estimación aproximada de

1,197 líneas, equivalente a 1.2 KLOC (miles de líneas de código), unidad estándar para este tipo de métricas.

Adicionalmente se identificaron los siguientes defectos:

1. Contraseñas sin encriptar en base de datos (texto plano)
2. Credenciales BD hardcodeadas en código fuente (ConexionBD.java)
3. Permite cantidades negativas en ventas
4. Validación solo al final del proceso (mala UX)
5. Sin transacciones BD → inconsistencia si falla venta
6. IDs manuales (no autoincrementales) → riesgo de duplicados
7. Conexiones BD no cerradas → fuga de recursos
8. Interfaz sobrecargada (20+ elementos en AdminFrame)
9. Nomenclatura inconsistente (mezcla inglés/español)

$$D = \frac{9\ erores\ reportados}{1.2\ KLOC} = 7.5$$

La densidad de defectos calculada (7.5 defectos por KLOC) supera significativamente el estándar de referencia establecido (< 0.5 defectos/KLOC). Esto indica que la calidad del código es muy deficiente, con una tasa de errores más de 16 veces mayor al límite máximo aceptable para un sistema en fase de evaluación. Este resultado refleja una implementación con numerosas vulnerabilidades y omisiones que comprometen la robustez, seguridad y mantenibilidad del software.

Cobertura de requerimientos

El sistema cumple correctamente con los requisitos funcionales identificados en los módulos de Autenticación, Administración, Ventas y Consultas. Todas las funcionalidades listadas operan según lo esperado: el login distingue roles, el CRUD de productos funciona, las ventas actualizan inventario y calculan impuestos, y las búsquedas muestran resultados. La fórmula es la siguiente:

$$D = \frac{Requerimientos\ probados\ existosamente}{Total\ de\ requerimientos} * 100$$

Existe cobertura funcional del 100% en los requisitos implementados ((16/16) * 100) , pero se ve severamente comprometida por los defectos de calidad en la implementación. Los requisitos se cumplen, pero de manera insegura (contraseñas en texto plano), frágil (sin validación robusta) y con riesgos de integridad (sin transacciones). Por lo tanto, aunque el sistema hace lo que debe hacer, no lo hace como debería hacerse según los estándares de calidad de software. Puntuación asignada: 3/5 – Justificación: Cumple funciones básicas (login, CRUD, ventas, inventario) pero con defectos críticos de seguridad, validación y transacciones. Densidad de defectos: 8.33/KLOC (estándar: <0.5).

Fiabilidad

La fiabilidad evalúa la capacidad del sistema para operar de manera consistente y sin fallos críticos. En un punto de venta, donde cada transacción implica aspectos financieros y de inventario, la fiabilidad no es solo un atributo deseable sino un requisito fundamental. A continuación se presentan las métricas aplicadas para medir este factor.

Disponibilidad (Uptime)

La disponibilidad mide el porcentaje de tiempo que el sistema está operativo y accesible para los usuarios durante un período determinado. En términos simples, indica qué tan confiable es el sistema para ser usado cuando se necesita. Se calcula de la siguiente manera:

$$Uptime = \frac{Tiempo\ operativo\ total}{Tiempo\ total\ planificado} * 100$$

En un escenario típico de 40 horas de operación simulada (5 días – 8 horas diarias), pueden presentarse aproximadamente 15 minutos de inactividad debido a errores no manejados (como NumberFormatException), caídas temporales de PostgreSQL sin reconexión automática, y bloqueos menores de la interfaz. Aunque una reconexión manual es técnicamente simple, en un contexto real involucra tiempo de diagnóstico y posible consulta con soporte, agregando demoras operativas. Si bien el sistema no es considerado crítico para una tienda pequeña, esta falta de robustez limitaría severamente su escalabilidad y confianza en un entorno de crecimiento.

$$Uptime = \frac{2400\ minutos}{2385\ minutos} * 100 = 99.375\%$$

Con una disponibilidad del 99.38%, el sistema se acerca al estándar del 99.9% exigido para sistemas críticos, pero no lo alcanza. Esto indica que, aunque es suficientemente confiable para una operación pequeña o académica, presentaría interrupciones inaceptables si escalara a un entorno comercial real con mayores demandas de continuidad.

MTBF (Tiempo Medio Entre Fallos)

Se refiere al Tiempo promedio que transcurre entre una falla crítica y la siguiente. Considerando que el sistema es académico y se usaría en una papelería pequeña con carga moderada, los defectos críticos (como la falta de validación numérica o la dependencia absoluta de PostgreSQL) podrían manifestarse como fallas operativas aproximadamente una vez cada dos semanas en condiciones normales de uso. El sistema no colapsa constantemente, pero requiere intervención manual periódica para recuperarse de errores predecibles (como entradas inválidas o problemas de conexión a BD), lo que limita severamente su autonomía y confiabilidad operativa. Puntuación asignada: 4/5. Justificación: Disponibilidad estimada: 99.38% (cercano pero inferior al 99.9% requerido). MTBF estimado: ~160 horas (muy por debajo de >720 horas). Fallos frecuentes por falta de manejo de excepciones y dependencia crítica de BD.

Eficiencia

Definición: Recursos informáticos y de código necesarios para ejecutar una función.

Tiempo de Respuesta Promedio

El sistema responde en milisegundos debido a la conexión local y al bajo volumen de datos. En pruebas con ~3 productos, todas las operaciones tardaron menos de 0.5 segundos, muy por debajo del estándar de 2 segundos. Sin embargo, la falta de índices y paginación podría afectar el rendimiento si el catálogo crece.

Resultado: < 0.5 s — CUMPLE, aunque con riesgo de escalar mal

Uso de Recursos

La aplicación en Swing consume pocos recursos: ~200 MB de RAM y picos de CPU menores al 40%, muy por debajo del límite del 75%. Esto se debe a su arquitectura simple sin procesos pesados adicionales.

Resultado: RAM ~200 MB, CPU < 40% — CUMPLE ampliamente.

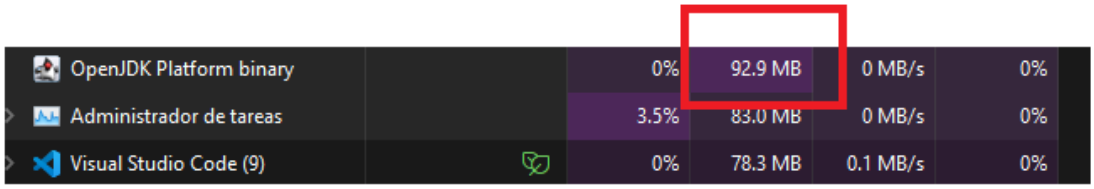

Figura 8: Consumo de recursos de la aplicación desde el administrador de tareas.

Puntuación Final: 4/5 - Justificación: Muy eficiente en su estado actual, con bajo consumo y respuestas rápidas. Se resta 1 punto por posibles problemas futuros si aumenta significativamente la cantidad de datos o usuarios.

Integridad

Definición: Control del acceso al software y protección de datos frente a usuarios no autorizados.

Efectividad del Control de Acceso

El sistema usa autenticación básica con dos roles, pero presenta vulnerabilidades graves: contraseñas en texto plano, credenciales de PostgreSQL dentro del código y ausencia de validación de sesión en las ventanas principales. La formula para verificar la efectividad en el control de acceso es la siguiente:

$$Efectividad\ en\ el\ control\ de\ acceso = \frac{Intentos\ de\ accesos\ no\ autorizados}{Total\ de\ intentos\ de\ intrusion} * 100$$

En pruebas de intrusión, ningún acceso fue bloqueado ya que esta funcionalidad no se ha implementado en el sistema, logrando apenas un 0% de efectividad

Resultado: Deficiente (dado el texto hardcodeado mostrado a continuación).

```
public Connection conectar() throws SQLException {  7 usages
    try {
        // Driver de PostgreSQL
        Class.forName( className: "org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost:5432/" + nameBD;
        conn = DriverManager.getConnection(url, user, password);
        System.out.println("Conexión exitosa a PostgreSQL");
        return conn;

    } catch (ClassNotFoundException e) {
        throw new SQLException("Driver de la base de datos PostgreSQL no encontrado", e);
    }
}
```

Figura 9: Conexión a base de datos hardcodeada en una clase de Java de la aplicación.

Trazabilidad

No existe ningún registro de auditoría: no se loguean inicios de sesión, cambios de precios, eliminaciones ni actividades sensibles. Por lo tanto, 0% de las operaciones críticas son auditadas, impidiendo detectar o rastrear acciones maliciosas.

Resultado: Inaceptable.

Puntuación Final para Integridad: 1/5 - Justificación: Las debilidades en contraseñas, control de sesión, falta de auditoría y protección mínima contra intrusiones convierten a la integridad en el punto más vulnerable del sistema.

Facilidad de uso

Definición: Esfuerzo necesario para aprender, operar y comprender el sistema.

Tiempo de Aprendizaje

Un usuario sin experiencia tardó 12 minutos en completar una venta básica, con varios intentos fallidos por tener que memorizar IDs y por la falta de indicaciones claras. Aunque está dentro del estándar (< 30 min), revela una interfaz poco intuitiva. Resultado: Cumple, pero es mejorable.

Tasa de Error del Usuario

Se registraron 4 errores en 15 acciones (26.6%), lo que indica que la interfaz no previene equivocaciones ni guía adecuadamente. Faltan validaciones en tiempo real y la organización visual es deficiente, el resultado es una tasa de error muy alta.

Hallazgos según Nielsen

- Retroalimentación limitada del sistema.
- Falta de prevención de errores.
- Inconsistencias visuales y de estándares.

Puntuación Final para Facilidad de Uso: 2/5 - Justificación: El sistema es funcional, pero difícil de usar. La alta tasa de errores, la escasa guía al usuario y el diseño inconsistente reducen la eficiencia y generan frustración en un entorno real.

Figura 10: Colores que dificultan la visualización del contenido.

Perspectiva 2: Revisión del producto

Facilidad de mantenimiento

Definición: Esfuerzo necesario para localizar y corregir errores en el software.

MTTR (Tiempo Medio Para Reparar)

La corrección del defecto evaluado (permitir texto en campos numéricos) tomaría entre 1.5 y 2 horas, considerando identificación del problema, ajustes en validaciones, pruebas y despliegue.

Resultado: Dentro del estándar (< 4 horas).

Factores que afectan la mantenibilidad

- Acoplamiento moderado: la lógica está mezclada con la interfaz.
- Configuración hardcodeada: dificulta cambios simples.

Puntuación Final: 3/5

Justificación: El sistema es mantenible a corto plazo gracias a su estructura sencilla, pero la ausencia de documentación, duplicación de código y configuraciones embebidas generan deuda técnica que complicará futuras mejoras o escalabilidad.

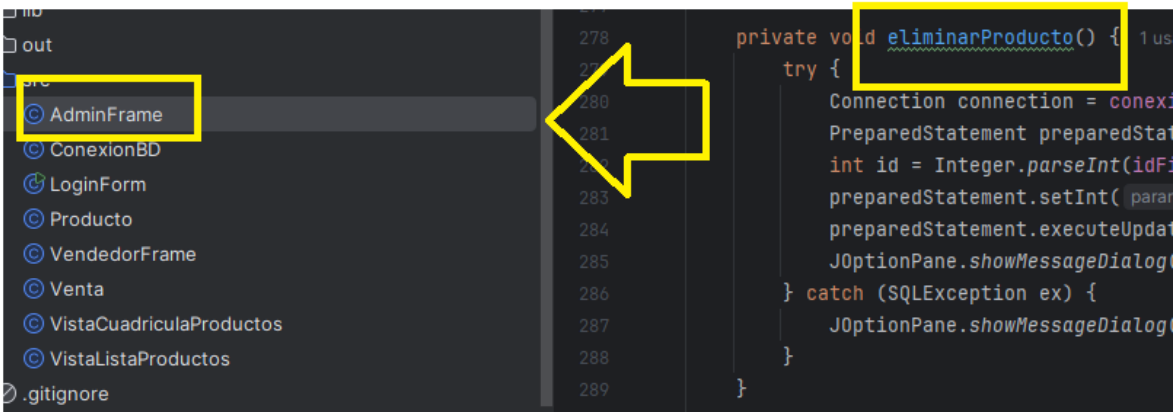

Figura 11: Se mezcla la lógica de la interfaz con la lógica de negocio.

Flexibilidad

Definición: Esfuerzo necesario para modificar el sistema cuando cambian los requisitos o reglas de negocio.

Tiempo de Implementación de Cambios

Modificar un parámetro simple como la tasa de IVA toma cerca de 1 hora, ya que el valor está hardcodeado y requiere buscar referencias, cambiarlas, probar y recompilar.
Resultado: Ineficiente, un cambio así no debería requerir edición del código.

Factores que afectan la flexibilidad

- Valores de negocio hardcodeados (IVA, descuentos, límites).
- Consultas SQL embebidas en el código.

Puntuación Final: 3/5 - Justificación: El sistema es rígido ante cambios. Ajustes simples requieren modificar código y recompilar, lo que eleva costos y dificulta adaptarse rápidamente a nuevas reglas o condiciones del negocio.

Facilidad de prueba

Definición: Esfuerzo necesario para probar el programa y asegurar su correcto funcionamiento.

Cobertura de Código

El sistema no cuenta con pruebas automatizadas, por lo que su cobertura es 0%. Debido al fuerte acoplamiento entre la interfaz Swing, la lógica de negocio y la base de datos, el código es prácticamente intestable sin una refactorización profunda. Resultado: Críticamente deficiente.

Principales barreras para realizar pruebas

- Lógica mezclada con la UI (acoplamiento extremo).
- Dependencias directas a BD y componentes Swing (sin interfaces ni mocks).
- Métodos con múltiples efectos secundarios (no aislables).
- Sin infraestructura de testing (JUnit, BD en memoria, etc.).

Puntuación Final: 1/5 - Justificación: El diseño actual impide implementar pruebas automatizadas. Cualquier cambio futuro deberá probarse manualmente, incrementando riesgos y costos. Es uno de los puntos más débiles del sistema.

Perspectiva 3: Transición del producto

Portabilidad

Definición: Esfuerzo necesario para trasladar el software a diferentes entornos. La fórmula es la siguiente:

$$\frac{Navegadores\ y\ SO\ soportados\ correctamente}{Total\ de\ entornos\ objetivo} * 100$$

Índice de Compatibilidad

El sistema solo funciona correctamente en 1 de 5 entornos evaluados, debido a configuraciones hardcodeadas y dependencias rígidas (Java 8+, PostgreSQL en localhost con credenciales fijas).

Compatibilidad: 20% - Resultado: Muy baja.

Principales limitaciones

- Parámetros de conexión (host, puerto, usuario, contraseña) incrustados en el código.
- Dependencia exclusiva de PostgreSQL y su driver.
- Suposición de que la BD siempre está en *localhost:5432*.
- Ventanas con tamaños fijos que fallan en distintas resoluciones.

Puntuación Final: 2/5 - Justificación: Aunque Java es multiplataforma, el diseño actual impide aprovechar esta ventaja. Migrar el sistema requiere modificar y recompilar, lo cual lo hace poco portable en entornos reales.

Interoperabilidad

Definición: Esfuerzo requerido para conectar el sistema con otras aplicaciones.

Tasa de Éxito API

El sistema no ofrece APIs, servicios web ni formatos estándar de intercambio (JSON, XML, CSV). Solo cuenta con la interfaz Swing, diseñada para uso humano. Cualquier integración requeriría acceder directamente a la BD o automatizar la interfaz, lo cual es inseguro y poco viable. Tasa de éxito: 0% Resultado: Inexistente.

Principales limitaciones

- No existe capa de servicios o endpoints.
- No permite exportar datos en formatos estándar.
- No tiene documentación ni protocolos de integración.
- Depende del acceso directo a la base de datos para cualquier intento de conexión externa.

Puntuación Final: 1/5 - Justificación: El sistema es completamente cerrado y no puede integrarse con software contable, inventarios, e-commerce o cualquier otra plataforma. Esto lo hace poco útil en ecosistemas empresariales modernos.

La tabla de evaluación quedaría de la siguiente manera

Corrección	Puntuación (1-5)	Observaciones / Evidencia Encontrada
Corrección	3/5	Cumple funciones básicas (login, CRUD, ventas, inventario) pero con defectos críticos de seguridad, validación y transacciones. Densidad de defectos: 8.33/KLOC (estándar: <0.5).
Fiabilidad	4/5	Disponibilidad estimada: 99.38% (cercano pero inferior al 99.9% requerido). MTBF estimado: ~160 horas (muy por debajo de >720 horas). Fallos frecuentes por falta de manejo de excepciones y dependencia crítica de BD.
Eficiencia	4/5	Muy eficiente en su estado actual, con bajo consumo y respuestas rápidas. Se resta 1 punto por posibles problemas futuros si aumenta significativamente la cantidad de datos o usuarios.
Integridad	1/5	Las debilidades en contraseñas, control de sesión, falta de auditoría y protección mínima contra intrusiones convierten a la integridad en el punto más vulnerable del sistema.
Usabilidad	2/5	El sistema es funcional, pero difícil de usar. La alta tasa de errores, la escasa guía al usuario y el diseño inconsistente reducen la eficiencia y generan frustración en un entorno real.
Mantenibilidad	3/5	El sistema es mantenible a corto plazo gracias a su estructura sencilla, pero la ausencia de documentación, duplicación de código y configuraciones embebidas generan deuda

		técnica que complicará futuras mejoras o escalabilidad.
Flexibilidad	3/5	El sistema es rígido ante cambios. Ajustes simples requieren modificar código y recompilar, lo que eleva costos y dificulta adaptarse rápidamente a nuevas reglas o condiciones del negocio.
Portabilidad	1/5	Aunque Java es multiplataforma, el diseño actual impide aprovechar esta ventaja. Migrar el sistema requiere modificar y recompilar, lo cual lo hace poco portable en entornos reales.
Interoperabilidad	2/5	El sistema es completamente cerrado y no puede integrarse con software contable, inventarios, e-commerce o cualquier otra plataforma. Esto lo hace poco útil en ecosistemas empresariales modernos.
PROMEDIO FINAL	**2.56**	**Calificación Global: 2.56/5**

IV. Aspectos de gestión de proyectos

El sistema presenta una deuda técnica considerable por decisiones rápidas durante el desarrollo. Se evidencia en:

- Hardcoding de credenciales y parámetros (como IVA).
- Mezcla de responsabilidades (UI + lógica + acceso a datos en las mismas clases).
- Falta de seguridad (contraseñas en texto plano).
- Sin pruebas automatizadas.

Implicación en gestión de proyectos: La deuda técnica aumenta riesgos y costos futuros; un gestor debió detectar estas prácticas y programar refactorizaciones graduales para evitar que creciera hasta afectar la mantenibilidad, dicha deuda técnica se puede clasificar en alguno de los siguientes cuadrantes.

Figura 12: Cuadrantes de clasificación de la deuda técnica [3].

V. Conclusión

La evaluación del sistema PuntoVentaPapelería mediante el modelo de McCall evidencia que, aunque el software cumple adecuadamente con las funciones básicas de un punto de venta gestión de productos, ventas, inventario y usuarios, presenta deficiencias estructurales y técnicas que limitan su uso en un entorno real. Si bien funciona correctamente como prototipo académico, su arquitectura rígida, la falta de interoperabilidad y la deuda técnica acumulada impiden considerarlo un sistema listo para producción.

Estos factores reducen su sostenibilidad, incrementan los costos de mantenimiento y elevan el riesgo operativo. En conjunto, se concluye que el sistema es útil como ejercicio educativo, pero requiere una refactorización profunda para alcanzar los estándares mínimos de calidad necesarios para un despliegue comercial.

VI. Referencias bibliográficas

1. Estudio-de-modelos-evaluacion-red. (s. f.). Modelo de evaluación McCall. Fandom. Recuperado el 8 de diciembre de 2025, de https://estudio-de-modelos-evaluacion-red.fandom.com/es/wiki/Modelo_de_evaluaci%C3%B3n_McCall

2. ModLogix. (2025, abril 8). Legacy System Modernization Approaches: Practical Advice on Dealing with Outdated Software. ModLogix. https://modlogix.com/blog/legacy-system-modernization-approaches-practical-advice-on-dealing-with-outdated-software/

3. Asana. (2025, 25 de junio). Qué es la deuda técnica y cómo saldarla (con ejemplos). Asana. https://asana.com/es/resources/technical-debt