Tarea Unidad 1: Diseño de una aplicación para una tienda especializada en vender productos estéticos

Entornos de desarrollo

Curso DAM 2024/25

Calificación obtenida: 10/10

ÍNDICE

1. Sintetiza el análisis de requerimientos del sistema para nuestro cliente. Plantea el diseño y determina el modelo de ciclo de vida más idóneo para esta aplicación. 3
1.1. Especificación de requisitos software 3
1.2. Diseño de la arquitectura 4
● Estructura relacional global del sistema 4
● Módulos 4
● Plan de pruebas 5
1.3. Modelo de ciclo de vida 6
2. Planifica la codificación, indicando el lenguaje de programación y las herramientas que usarías para la obtención del código fuente, objeto y ejecutable, explicando por qué eliges esas herramientas. 7
2.1. Lenguaje de programación 7
2.2. Herramientas para la obtención del código fuente 8
2.2.1. Código fuente 8
2.2.2. Objeto 8
2.2.3. Ejecutable 8
3. Planifica las restantes fases del ciclo de vida, indicando en cada una el objetivo que persigues y cómo lo harías. 9
3.1. Fase de Pruebas 9
Pruebas Unitarias 9
Pruebas de Integración 9
Otras pruebas 10
3.2. Fase de explotación 10
Instalación de la Aplicación 10
Puesta a Punto 10
Funcionamiento de la Aplicación 11
Capacitación y Documentación 11
Recopilación de Feedback 11
3.3. Fase de mantenimiento 11
3.4. Fase de documentación 12
4. Indica el ciclo de vida que usarías. 13

ENUNCIADO

La empresa BK ha recibido un nuevo encargo de software.

Se trata de diseñar una aplicación para una tienda especializada en vender productos estéticos.

La tienda desea trabajar con software libre. Además, desea explícitamente que la aplicación sea capaz de cumplir las siguientes tareas:

1. Proporcionar facturas de las ventas.
2. Llevar la cuenta de lo que vende cada trabajador.
3. Controlar el stock de productos en almacén.
4. Operar con lector de código de barras y tarjetas de crédito.
5. Controlar los precios de los productos y ofrecer la posibilidad de operar con ellos.
6. El tiempo de respuesta de la aplicación ha de ser lo menor posible.
7. No se podrán procesar dos peticiones a la vez, aunque haya varios equipos funcionando simultáneamente.
8. La empresa también quiere almacenar información de sus trabajadores: DNI, nombre, apellidos, número de la Seguridad Social, fecha de nacimiento, teléfono y localidad. Asimismo, de los productos interesa almacenar: código, marca, nombre comercial, precio, cantidad.

1. Sintetiza el análisis de requerimientos del sistema para nuestro cliente. Plantea el diseño y determina el modelo de ciclo de vida más idóneo para esta aplicación.

1.1. Especificación de requisitos software

REQUISITOS FUNCIONALES	REQUISITOS NO FUNCIONALES
Proporcionar facturas de las ventas	Software libre
Llevar la cuenta de lo que vende cada trabajador	El tiempo de respuesta de la aplicación ha de ser lo menor posible
Controlar el stock de productos en almacén	No se podrán procesar dos peticiones a la vez, aunque haya varios equipos funcionando simultáneamente
Operar con lector de código de barras y tarjetas de crédito	La información de los trabajadores y productos debe ser almacenada de forma segura.
Controlar los precios de los productos y ofrecer la posibilidad de operar con ellos	

Almacenamiento de información de trabajadores (DNI, nombre, apellidos, número de la Seguridad Social, fecha de nacimiento, teléfono y localidad)	
Almacenamiento de datos de productos (código, marca, nombre comercial, precio, cantidad)	

1.2. Diseño de la arquitectura

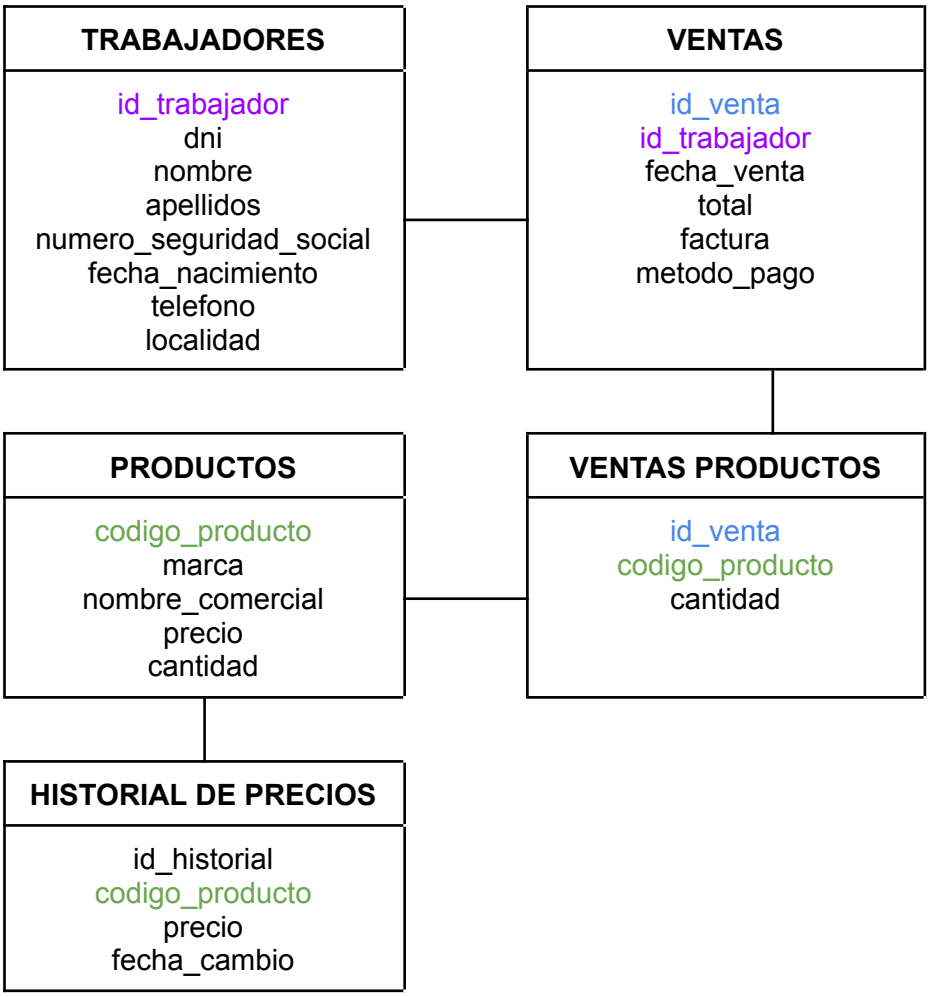
- **Estructura relacional global del sistema**

    - **Cliente-servidor:** La aplicación se puede diseñar en arquitectura cliente-servidor, donde varias terminales (clientes) acceden a un servidor central para manejar el inventario, los datos de ventas y el procesamiento de pagos.
    - **Base de datos centralizada:** El servidor tendrá una base de datos central que manejará los datos de productos, trabajadores y ventas.
    - **Interfaz gráfica de usuario:** Aplicación usada por los trabajadores

- **Módulos**

REQUISITOS FUNCIONALES	MÓDULOS
Almacenamiento de datos de productos (código, marca, nombre comercial, precio, cantidad)	Gestión de productos y stock (PRODUCTOS): Un módulo para el almacenamiento de los productos y el control del stock.
Controlar el stock de productos en almacén	Gestionar los productos vendidos (VENTAS_PRODUCTOS): Relaciona las ventas con los productos vendidos, permitiendo ventas de múltiples productos.
Controlar los precios de los productos y ofrecer la posibilidad de operar con ellos	Control de precios (HISTORIAL DE PRECIOS): Funcionalidad para gestionar los precios de los productos
Operar con lector de código de barras y tarjetas de crédito	La integración con los periféricos (lectores de código de barras y tarjetas de crédito) sería manejada a nivel de aplicación, no en la estructura de la base de datos.
Proporcionar facturas de las ventas	Facturación (VENTAS): Módulo que genera facturas de manera automática después de cada venta

Llevar la cuenta de lo que vende cada trabajador	Ventas por trabajador (VENTAS): Un módulo que permite que las ventas se asocien a un trabajador específico y muestra un resumen de sus ventas
Almacenamiento de información de trabajadores (DNI, nombre, apellidos, número de la Seguridad Social, fecha de nacimiento, teléfono y localidad)	Datos trabajadores (TRABAJADORES): Un módulo para almacenar datos de los trabajadores



- ## Plan de pruebas

El plan de pruebas tiene como objetivo definir la estrategia y el enfoque para validar y verificar que la aplicación desarrollada para la tienda de productos estéticos cumple con los requisitos especificados y funciona correctamente.

Objetivos de las Pruebas:

- Asegurar que cada módulo de la aplicación funcione correctamente de manera independiente (pruebas unitarias). Como herramienta se usará JUnit. Criterios de Éxito: Todas las pruebas unitarias deben pasar sin errores.

- Verificar que los módulos interactúan correctamente entre sí (pruebas de integración). Como herramienta se usará NetBeans. Criterios de Éxito: Todas las interacciones entre módulos deben funcionar correctamente y los datos deben fluir sin errores.
- Validar que la aplicación cumple con los requisitos funcionales y no funcionales especificados.
- Identificar y corregir errores antes de la implementación final.

Documentación de Pruebas:
Se generará un documento de procedimiento de pruebas que incluirá:
- Detalles de las pruebas realizadas.
- Resultados obtenidos y comparación con los resultados esperados.
- Registro de errores encontrados y las acciones correctivas tomadas.

1.3. Modelo de ciclo de vida

El modelo de ciclo de vida más adecuado para este proyecto podría ser el **Modelo Iterativo Incremental**, por las siguientes razones:

Primero, este modelo **es muy flexible cuando se trata de hacer cambios.** A diferencia del modelo en cascada, que es bastante rígido y no deja hacer modificaciones una vez que se avanza, el modelo iterativo permite hacer entregas frecuentes. Esto facilita que se puedan ajustar los requisitos según lo que el cliente necesite. Aunque el modelo en espiral también tiene algo de flexibilidad, su enfoque en gestionar riesgos puede complicar las cosas y hacer que los costos suban. Por otro lado, el modelo ágil es flexible, pero requiere que el cliente esté muy involucrado, lo cual puede ser un problema si no puede comprometerse del todo.

Otra ventaja del modelo iterativo incremental es que **permite trabajar en partes y priorizar las funcionalidades**. Podemos dividir el proyecto en módulos y enfocarnos primero en las características más importantes, asegurando que el cliente empiece a ver resultados desde el principio. En cambio, el modelo en cascada sigue un orden estricto y no permite priorizar bien. Aunque el modelo en espiral permite cierta división del trabajo, su complejidad puede hacer que priorizar sea un poco más complicado. El modelo ágil sí permite priorizar, pero a veces le falta la estructura necesaria para que todo se desarrolle de manera ordenada.

Además, el modelo iterativo incremental permite **realizar pruebas continuas y mejorar constantemente**. Al realizar pruebas en cada iteración, se pueden identificar y solucionar problemas desde el principio, lo que ayuda a que el producto final sea de mejor calidad. En el modelo en cascada, las pruebas se hacen al final, lo que puede llevar a que se encuentren errores importantes muy tarde en el proceso. Aunque el modelo en espiral también permite hacer pruebas, su complejidad puede hacer que sea más difícil llevarlas a cabo de manera eficiente. El modelo ágil promueve hacer pruebas frecuentes, pero a veces no tiene la documentación y el seguimiento necesarios para asegurarse de que todo se pruebe como se debe.

Finalmente, el modelo iterativo incremental destaca por su **rapidez para adaptarse a cambios.** Permite ajustar los requisitos sin que eso afecte demasiado el proyecto, algo que

es complicado en el modelo en cascada, donde los cambios en etapas avanzadas pueden ser un gran problema. Aunque el modelo en espiral permite cierta adaptación, su enfoque puede hacer que responder a cambios sea más complicado. Y el modelo ágil, aunque es adaptable, depende de que el cliente esté muy presente, lo que puede limitar la rapidez con la que se responda a los cambios si el cliente no está disponible.

2. Planifica la codificación, indicando el lenguaje de programación y las herramientas que usarías para la obtención del código fuente, objeto y ejecutable, explicando por qué eliges esas herramientas.

2.1. Lenguaje de programación

Elijo **Java** como lenguaje orientado a objetos debido a varias ventajas frente a lenguajes estructurados y visuales, las cuales lo hacen más adecuado para este tipo de proyecto:

Java permite organizar el código de forma estructurada mediante clases y objetos. Cada parte del sistema (como los productos, las ventas o los trabajadores) puede ser representada por una clase, lo que **permite organizar y aislar el código de manera modular**. Esta modularidad **facilita el mantenimiento**, ya que las modificaciones a una parte del sistema no afectan a las demás, y **se pueden añadir nuevas funcionalidades** sin reestructurar el código completo. En comparación, los lenguajes estructurados no permiten una organización tan flexible, lo que puede complicar el mantenimiento a medida que el proyecto crece. Y en cuanto a los lenguajes visuales, estos permiten crear aplicaciones rápidamente, pero no proporcionan el mismo nivel de control estructural y modular que ofrece Java.

Una de las mayores ventajas de Java como lenguaje orientado a objetos es la **capacidad de reutilizar el código** mediante la herencia y la creación de clases base. Por ejemplo, si tenemos una clase `Producto`, se pueden crear subclases como `ProductoDeBelleza` o `ProductoDeSalud`, heredando comportamientos comunes y añadiendo nuevos comportamientos específicos. En los lenguajes estructurados si se quiere reutilizar código, se debe duplicar o escribir funciones adicionales, lo que no solo incrementa el riesgo de errores, sino que también hace que el mantenimiento del sistema sea más complejo y costoso. En cuanto a los lenguajes visuales aunque se pueden crear módulos reutilizables, estos no tienen la flexibilidad ni la organización de las clases en Java. Esto puede llevar a un sistema menos escalable y más difícil de mantener a largo plazo.

Otra característica de Java es que **ofrece una gran seguridad** gracias a su encapsulación (permite ocultar los detalles internos de las clases y exponer sólo lo necesario a través de métodos específicos), lo que protege los datos sensibles. Aunque tanto en los lenguajes estructurados como en los visuales se pueden implementar mecanismos para proteger los datos, estos no tienen un enfoque tan robusto de encapsulación como en Java, lo que hace que la seguridad sea más difícil de implementar correctamente.

2.2. Herramientas para la obtención del código fuente

Para la obtención del código fuente, objeto y ejecutable en un proyecto de desarrollo utilizando Java, se requieren diversas herramientas que faciliten cada una de estas etapas del proceso de codificación. A continuación, explico las herramientas que utilizaría y las razones para elegirlas:

2.2.1. Código fuente

El código fuente es el conjunto de instrucciones escritas por el programador en un lenguaje de programación (en este caso, Java). Estos archivos tienen la **extensión ".java"**. Para escribir código fuente utilizaría la herramienta **NetBeans** ya que este IDE está diseñado especialmente para el desarrollo en Java, facilita mucho el proceso de desarrollo (ofrece un soporte completo para la escritura de código, la depuración, la compilación y la ejecución de programas Java) y tiene una interfaz simple y fácil de usar. Aunque Eclipse es otro IDE popular para Java, su interfaz puede resultar más compleja y menos amigable para quienes están comenzando (como yo). Otra opción podría ser usar IntelliJ IDEA. Aunque IntelliJ IDEA tiene una versión gratuita llamada IntelliJ IDEA Community Edition, que es de código abierto, algunas características avanzadas solo están disponibles en la versión de pago. Me decanto así por NetBeans porque es completamente software libre y de código abierto (que es uno de los requerimientos de nuestro proyecto).

2.2.2. Objeto

Para la fase de compilación, donde se obtiene el código objeto (los **archivos .class**), usaría la herramienta **JDK (Java Development Kit)**. El JDK incluye el compilador javac, que se utiliza para compilar el código fuente y generar el código objeto. El JDK es una herramienta estándar y ampliamente utilizada en la comunidad Java. El JDK, al ser software libre, me permite acceder a actualizaciones y mejoras sin costo. No uso otras alternativas que funcionan de manera más automática, como Maven o Gradle, porque en principio el proyecto que tenemos entre manos es pequeño y creo que no necesita de herramientas tan complejas. Además, si en un futuro el proyecto se complicase, es posible cambiar de JDK a una de estas dos herramientas.

2.2.3. Ejecutable

Para la generación del código ejecutable (el **archivo .jar** que contendrá el código compilado), utilizaré **JRE (Java Runtime Environment)** que es la parte del JDK que permite ejecutar aplicaciones Java. Al igual que el JDK, el JRE es software libre y está diseñado específicamente para ejecutar aplicaciones Java. Utilizar el JRE me asegura que su aplicación se ejecute en un entorno adecuado y optimizado para Java.

3. Planifica las restantes fases del ciclo de vida, indicando en cada una el objetivo que persigues y cómo lo harías.

3.1. Fase de Pruebas

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas. Esta etapa es crucial para asegurar la validación y verificación del software construido, garantizando que cumple con los requisitos especificados y funciona correctamente en diferentes escenarios.

En el contexto del desarrollo de la aplicación para la tienda especializada en productos estéticos, se llevarán a cabo principalmente dos tipos de pruebas: pruebas unitarias y pruebas de integración.

Pruebas Unitarias

Las pruebas unitarias se enfocan en verificar el funcionamiento de las diferentes partes del software de manera individual. Cada módulo o componente del sistema se prueba por separado para asegurar que cada uno funcione correctamente de acuerdo con sus especificaciones.

Se desarrollará un conjunto de casos de prueba para cada módulo, basándose en el plan de pruebas establecido durante la fase de diseño.

Se utilizará **JUnit**, un entorno de pruebas unitarias para Java, que permite crear y ejecutar pruebas de manera eficiente.

Cada prueba se ejecutará y los resultados obtenidos se compararán con los resultados esperados, que se habrán definido previamente.

Se generará un documento de procedimiento de pruebas que incluirá los resultados de las pruebas unitarias, así como cualquier error encontrado y las correcciones realizadas.

Pruebas de Integración

Las pruebas de integración se llevan a cabo después de que todas las partes individuales del software han pasado las pruebas unitarias. Esta fase consiste en poner en común todos los módulos desarrollados y verificar que interactúan correctamente entre sí.

Se desarrollará un plan de pruebas de integración que describa cómo se van a probar las interacciones entre los diferentes módulos.

Se ejecutarán pruebas que simulen escenarios reales de uso, asegurando que los módulos se comuniquen correctamente y que los datos fluyan como se espera a través del sistema.

Al igual que en las pruebas unitarias, se generará un documento de procedimiento de pruebas de integración que documentará los resultados, comparando los resultados obtenidos con los resultados esperados.

Se abordará cualquier problema de integración que surja, ajustando el código según sea necesario para resolver los conflictos entre módulos.

Otras pruebas

- **Pruebas funcionales:** Realizar pruebas funcionales para comprobar que todas las funcionalidades requeridas se implementan y funcionan como se espera.
- **Pruebas de rendimiento:** Ejecutar pruebas de rendimiento para garantizar que el tiempo de respuesta de la aplicación sea el menor posible.
- **Pruebas con el cliente:** Involucrar al cliente en pruebas de aceptación para asegurar que el producto final cumple con sus expectativas.

3.2. Fase de explotación

La fase de explotación se refiere a la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente. Esta etapa es crucial para asegurar que la aplicación opere de manera efectiva y cumpla con las expectativas del usuario final. A continuación, se detalla el proceso de explotación:

Instalación de la Aplicación

- Preparación del Entorno del Cliente: Verificar que el equipo del cliente cumple con los requisitos técnicos necesarios para la instalación de la aplicación, incluyendo la versión adecuada del sistema operativo y la instalación del JRE (Java Runtime Environment).
- Instalación del Archivo Ejecutable: Transferir el archivo ejecutable .jar de la aplicación al equipo del cliente.
- Ejecutar el archivo .jar para iniciar la instalación y asegurarse de que todos los componentes necesarios se configuran adecuadamente en el sistema del cliente.

Puesta a Punto

- Configuración Inicial: Realizar la configuración inicial de la aplicación, incluida la conexión a la base de datos, la configuración de parámetros específicos del cliente y la personalización de la interfaz de usuario según las preferencias del usuario final.
- Pruebas de Funcionamiento: Antes de la implementación completa, realizar una fase de **Beta Testing** donde un grupo selecto de usuarios finales (por ejemplo, empleados de la tienda) utilizarán la aplicación en un entorno real durante un período determinado.
- Recoger feedback y observaciones sobre la funcionalidad, usabilidad y cualquier problema encontrado durante el uso de la aplicación.

Funcionamiento de la Aplicación

- Monitoreo del Rendimiento: Durante las primeras semanas de funcionamiento, realizar un seguimiento del rendimiento de la aplicación para asegurarse de que funcione sin problemas y cumpla con los requisitos de rendimiento establecidos.
- Soporte Técnico: Proporcionar soporte técnico al cliente para resolver cualquier inconveniente que pueda surgir durante el uso diario de la aplicación.
- Establecer un canal de comunicación (como un correo electrónico o un número de teléfono) para que los usuarios puedan reportar problemas o hacer preguntas.

Capacitación y Documentación

- Capacitación Adicional: Ofrecer sesiones de capacitación adicionales si es necesario, para asegurar que todos los usuarios estén cómodos y familiarizados con el uso de la aplicación.
- Documentación de Soporte: Proporcionar documentación de soporte, como guías de usuario y manuales de solución de problemas, que los usuarios puedan consultar en caso de dudas o dificultades.

Recopilación de Feedback

- Evaluación del Usuario Final: Recoger feedback de los usuarios sobre su experiencia con la aplicación, identificando áreas de mejora y posibles nuevas funcionalidades que podrían ser útiles.
- Ajustes y Mejoras: Basándose en el feedback recibido, planificar y desarrollar ajustes o mejoras que puedan implementarse en futuras actualizaciones de la aplicación.

3.3. Fase de mantenimiento

La fase de mantenimiento asegura que la aplicación siga siendo útil y efectiva a lo largo de su vida útil. Por ello se pacta con el cliente un servicio de mantenimiento, que incluye un coste tanto temporal como económico.

El Plan de mantenimiento incluirá aspectos:

- **Perfectivos:** Para mejorar la funcionalidad del software.
- **Evolutivos:** El cliente propone mejoras para el producto. Implica nuevos requisitos.
- **Adaptativos:** Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, nuevas condiciones especificadas por organismos reguladores, etc.
- **Correctivos:** Resolver errores detectados.

Para ello es necesario establecer un sistema de monitoreo para detectar problemas de rendimiento y errores en tiempo real; realizar evaluaciones periódicas del software para identificar áreas de mejora y posibles actualizaciones; y mantener una comunicación constante con el cliente para comprender sus necesidades y expectativas, lo que permitirá adaptar el software a sus requerimientos en evolución.

3.4. Fase de documentación

La elaboración de documentos es constante durante todo el ciclo de vida. De acuerdo al destinatario final de los documentos, obtendremos tres tipos documentos:

- **Documentos dirigidos al personal técnico en informática** (analistas y programadores): Estos documentos están orientados a facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro. Incluyen aspectos relacionados con el diseño de la aplicación, la codificación de los programas y las pruebas realizadas
- **Documentos dirigidos a los usuarios que van a usar la aplicación (clientes):** Dan a los usuarios finales toda la información necesaria para utilizar la aplicación.Incluyen información relacionada con la descripción de la funcionalidad de la aplicación, la forma de comenzar a ejecutar la aplicación, ejemplos de uso del programa, requerimientos de software de la aplicación y la solución de los posibles problemas que se pueden presentar.
- **Documentos dirigidos al personal informático responsable de la instalación:** Dan toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa. Contienen información referente a la puesta en marcha, la explotación y la seguridad del sistema.

4. Indica el ciclo de vida que usarías.

	ANÁLISIS	DISEÑO	CODIFICACIÓN	PRUEBAS	EXPLOTACIÓN	MANTENIMIENTO	DOCUMENTACIÓN
	-Identificación de los requisitos (Funcionales y no funcionales)	-Descripción de la estructura relacional global del sistema - Especificación de lo que debe hacer cada una de sus partes - Manera en que se combinan unas partes con otras.					-Documento especificación de requisitos software -Documento de diseño de arquitectura

	ANÁLISIS	DISEÑO	CODIFICACIÓN	PRUEBAS	EXPLOTACIÓN	MANTENIMIENTO	DOCUMENTACIÓN
PRIMER INCREMENTO	-Definición del alcance	- Estructura de la base de datos - Conexión inicial desde Java	Implementación de la funcionalidad de facturación y control de stock	Pruebas iniciales de conectividad y funcionalidad básica	Implementación en entorno de desarrollo	Ajustes y corrección de errores encontrados	Documentación de la funcionalidad inicial y configuración del entorno

	ANÁLISIS	DISEÑO	CODIFICACIÓN	PRUEBAS	EXPLOTACIÓN	MANTENIMIENTO	DOCUMENTACIÓN
SEGUNDO INCREMENTO	Revisión de feedback inicial y planificación de funcionalidades avanzadas.	Estructura de interfaz avanzada y diseño de gestión de precios.	Integración de lector de códigos de barras y control de ventas por trabajador.	Validación de funcionalidades avanzadas y pruebas de rendimiento.	Implementación de funcionalidades avanzadas en entorno de producción	Ajustes basados en el uso real y retroalimentación.	Actualización de manuales de usuario y documentación técnica.

	ANÁLISIS	DISEÑO	CODIFICACIÓN	PRUEBAS	EXPLOTACIÓN	MANTENIMIENTO	DOCUMENTACIÓN
TERCER INCREMENTO	Evaluación final de requisitos adicionales y optimización.	Finalización de la interfaz y optimización de rendimiento.	Implementación de procesamiento de pagos con tarjeta de crédito y optimización final.	Pruebas de seguridad, integración final y pruebas de aceptación.	Despliegue completo de la aplicación en producción.	Supervisión del rendimiento y soporte técnico post-implementación.	Manuales de usuario finales y documentación completa del sistema.