INGENIERÍA DE SOFTWARE II

Universidad de la Cuenca del Plata · Posadas, Misiones · 2026

Informe de Trabajo Práctico N°1

Patrones de Diseño

Sistema de Gestión de Stock — Ferretería

Integrante	Rol
Santiago Gonzalez	Scrum Master
Luciano Fohrholtz	Dev Lead
Juan Carlos Abente	QA Lead
Mariano Acosta	UX Lead

Empresa ficticia: UCP Inc. · Materia: IS2 · Año: 2026

Sección 1

Introducción

El presente informe corresponde al Trabajo Práctico N°1 de la materia Ingeniería de Software II (IS2), dictada en la Universidad de la Cuenca del Plata (UCP), ciclo lectivo 2026. El trabajo fue desarrollado por el equipo UCP Inc., conformado por cuatro integrantes con roles diferenciados dentro de un marco metodológico Scrum.

El sistema desarrollado es Nipintucu, un sistema de gestión de stock para una ferretería mediana (Opción B de la consigna). El objetivo del TP1 es aplicar patrones de diseño del catálogo GoF dentro de funcionalidades reales del sistema, integrándolos de forma que resuelvan problemas concretos de diseño y puedan justificarse técnicamente.

1.1 Descripción del sistema

PROBLEMA	Una ferretería mediana necesita controlar su stock. Hoy no saben qué tienen ni cuándo pedir más. Quieren una solución que les avise cuando un producto está por agotarse.

USUARIOS	Empleado de mostrador: registra entradas y salidas de stock. Encargado de compras: consulta reportes y órdenes de reposición.

1.2 Funcionalidades implementadas

- Registrar productos con nombre, categoría, precio, stock actual y stock mínimo.
- Registrar entradas y salidas de stock con fecha, cantidad y motivo.
- Generar alertas automáticas cuando el stock cae por debajo del mínimo (patrón Observer).
- Generar reportes de productos a reponer y stock completo con algoritmo intercambiable (patrón Strategy).
- Buscar y filtrar productos por categoría.

1.3 Tecnologías utilizadas

Tecnología	Uso
Python 3.12	Lenguaje principal — clases del dominio y patrones
Flask	API REST para comunicación frontend-backend
HTML / CSS / JS	Interfaz web del sistema
GitHub Projects	Tablero Kanban — gestión del trabajo del equipo
Figma	Prototipo de interfaz de usuario (UX Lead)

Sección 2

Modelo de Dominio

El modelo de dominio fue construido a partir del análisis de las funcionalidades mínimas requeridas por la consigna. Se identificaron seis clases principales con sus atributos y relaciones.

2.1 Clases identificadas

Clase	Descripción
Producto	Entidad central. Contiene nombre, descripcion, precio, stock_actual y stock_minimo. Es el sujeto en el patrón Observer.
Categoria	Agrupa productos por rubro. Relación de agregación con Producto — existe aunque no tenga productos asignados.
StockMovimiento	Registra cada entrada o salida de stock con cantidad, tipo, fecha y motivo. Permite trazabilidad completa.
Alerta	Observador concreto. Se activa automáticamente cuando el stock de un Producto cae bajo el mínimo. Composición con Producto.

OrdenReposicion	Observador concreto. Genera una orden de reposición automática ante stock bajo.
Empleado	Actor del sistema. Registra movimientos de stock con nombre, apellido, legajo y rol.

2.2 Relaciones entre clases

Relación	Detalle
Composición	Producto compone Alerta — si se elimina el producto, la alerta no tiene existencia independiente.
Agregación	Categoria agrupa Producto — la categoría existe aunque no tenga productos asignados.
Asociación	Empleado registra StockMovimiento — el empleado existe independientemente de los movimientos.
Asociación	Producto genera StockMovimiento — cada movimiento referencia al producto afectado.

Sección 3

Patrones de Diseño Implementados

Se implementaron dos patrones de diseño del catálogo GoF, ambos de categoría Comportamental. Cada patrón fue seleccionado para resolver un problema concreto e identificado del sistema, no como ejemplo aislado.

3.1 Patrón Observer

CATEGORÍA	Comportamental — Catálogo GoF (Gamma, Helm, Johnson, Vlissides, 1994)

Intención

Definir una dependencia de uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

Problema identificado en el sistema

Cuando el stock de un Producto cae por debajo del stock_minimo, el sistema debe generar automáticamente una Alerta y una OrdenReposicion. Sin Observer, Producto tendría que

conocer y llamar directamente a esas clases, generando acoplamiento fuerte y violando el Principio de Responsabilidad Única (SRP) y el Principio Open/Closed (OCP).

Justificación de la elección

- **Desacoplamiento:** Producto desconoce qué hacen sus observadores. Solo conoce la interfaz abstracta Observador.
- **Extensibilidad (OCP):** Nuevos observadores se agregan sin modificar Producto (email, SMS, log en TP2).
- **Testeabilidad:** Producto puede testearse con observadores mock sin dependencias externas.
- **Responsabilidad única (SRP):** Cada clase tiene una única responsabilidad bien definida.

ALTERNATIVA DESCARTADA	Usar un método estático en una clase utilitaria NotificadorStock que Producto llamara directamente. Descartada porque seguía acoplando Producto a una implementación específica.

Estructura en el sistema

Rol	Clase / Descripción
Interfaz	Observador — define el método abstracto actualizar(producto)
Sujeto	Producto — mantiene la lista de observadores y llama a _notificar() al detectar stock bajo
Observador concreto 1	Alerta — registra la alerta en su historial interno
Observador concreto 2	OrdenReposicion — genera una orden de reposición automática
Trigger	Producto.registrar_salida() — descuenta stock y llama a _notificar() si stock_actual < stock_minimo

Mejoras que aporta

- **Desacoplamiento:** Producto y sus observadores están completamente desacoplados.
- **Mantenibilidad:** Agregar nuevos observadores en TP2 no requiere modificar Producto.
- **Claridad:** La lógica de notificación está centralizada en un solo método.

3.2 Patrón Strategy

CATEGORÍA	Comportamental — Catálogo GoF (Gamma, Helm, Johnson, Vlissides, 1994)

Intención

Definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. Strategy permite que el algoritmo varíe independientemente de los clientes que lo usan.

Problema identificado en el sistema

El sistema necesita generar distintos tipos de reportes: productos a reponer y stock completo. Sin Strategy, toda la lógica estaría mezclada en una sola clase con múltiples if/else, dificultando agregar nuevos tipos de reporte sin modificar código existente y violando el principio Open/Closed.

Justificación de la elección

- **Interfaz uniforme:** Cada reporte tiene la misma firma de método: recibe productos y devuelve un resultado estructurado.
- **Open/Closed (OCP):** Agregar un nuevo reporte en TP2 solo requiere crear una nueva clase, sin tocar GeneradorReporte.
- **Intercambiabilidad en runtime:** El tipo de reporte se selecciona en tiempo de ejecución según la acción del usuario.
- **Separación de responsabilidades:** Cada estrategia puede testearse de forma completamente independiente.

ALTERNATIVA DESCARTADA	Implementar los reportes como métodos distintos dentro de GeneradorReporte. Descartada porque viola SRP (múltiples razones de cambio) y OCP (agregar reporte requiere modificar la clase).

7 4

structura en el sistema

Rol	Clase / Descripción
Interfaz	EstrategiaReporte — define el método abstracto generar(productos)
Estrategia concreta 1	ReporteReposicion — filtra productos con stock_actual < stock_minimo

Estrategia concreta 2	ReporteStockActual — devuelve todos los productos con estado completo
Contexto	GeneradorReporte — recibe una estrategia y delega en ella la generación del reporte
Extensión TP2	ReporteMovimientosDiarios — nueva estrategia a agregar sin modificar código existente

Mejoras que aporta

- **Encapsulamiento:** Cada estrategia encapsula un algoritmo completo e independiente.
- **Mantenibilidad (OCP):** Nuevas estrategias se agregan sin modificar GeneradorReporte ni las existentes.
- **Flexibilidad en runtime:** El usuario selecciona el reporte en tiempo de ejecución desde la interfaz.

Sección 4

Caso de Uso Principal

CASO DE USO	Registrar una salida de stock y generar alerta si el stock queda bajo mínimo

Este caso de uso fue elegido como el núcleo del TP1 porque integra los dos patrones implementados (Observer y Strategy), cubre el flujo principal del sistema y puede ejecutarse de punta a punta con la implementación actual.

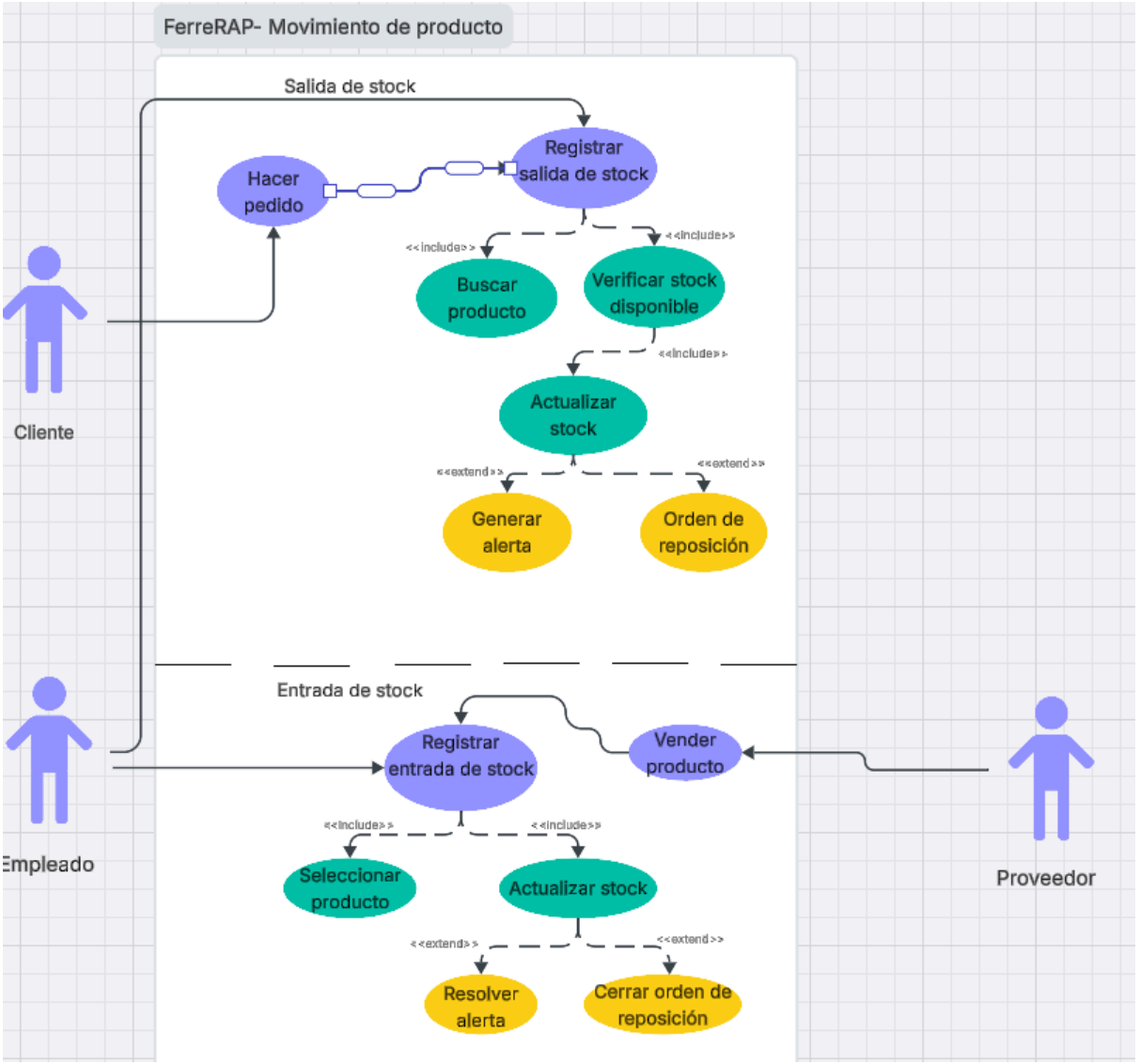
4.1 Flujo del caso de uso

1. El empleado accede al módulo de movimientos en la interfaz web.
2. Selecciona el producto, el tipo de movimiento (salida), la cantidad y el motivo.
3. El sistema ejecuta Producto.registrar_salida(cantidad, motivo).
4. Se descuenta la cantidad del stock_actual del producto.
5. Si stock_actual < stock_minimo, el patrón Observer notifica automáticamente a Alerta y OrdenReposicion.
6. La interfaz muestra el resultado del movimiento y las notificaciones generadas.
7. El empleado puede consultar el panel de Alertas para ver el historial.

4.2 Conexión con los patrones

Patrón	Rol en el caso de uso
Observer	Detecta automáticamente el stock bajo y notifica a los observadores registrados sin intervención del empleado.
Strategy	Permite al encargado de compras generar el reporte de productos a reponer con un clic, usando la estrategia ReporteReposicion.

4.3 Diagrama de caso de uso



Sección Salida de stock — el Cliente inicia la interacción (compra algo) y el Empleado ejecuta el proceso. El flujo incluye buscar el producto y verificar el stock disponible, que a su vez dispara la actualización. Si el stock cae por debajo del mínimo, se extiende condicionalmente a generar una alerta y crear una orden de reposición (el patrón Observer que ya tienen implementado).

Sección Entrada de stock — el Proveedor entrega mercadería y el Empleado la registra. Se incluye seleccionar el producto a reponer y actualizar el stock. Cuando el stock sube sobre el mínimo, se extienden condicionalmente los casos de resolver la alerta activa y cerrar la orden de reposición pendiente.
El Empleado aparece como actor en ambas secciones porque es quien opera el sistema en los dos flujos, mientras que Cliente y Proveedor son los actores externos que desencadenan cada movimiento.

Sección 5

Metodología y Gestión del Equipo

5.1 Metodología aplicada

El equipo adoptó Scrum como metodología de desarrollo. Se definieron sprints semanales con revisiones al inicio de cada clase. El Scrum Master (Santiago Gonzalez) fue responsable de mantener el tablero Kanban actualizado y de redactar el reporte semanal.

5.2 Tablero Kanban — estructura

Columna	Uso
Backlog	Tareas pendientes sin iniciar
En curso	Tareas activas de la semana
En revisión	Esperando feedback del docente o del equipo
Bloqueado	Impedimento documentado en la tarjeta
Hecho	Completado con evidencia (commit o link)

5.3 Roles del equipo

Integrante	Rol — Responsabilidad principal
Santiago Gonzalez	Scrum Master — organización, tablero Kanban, reportes semanales, documentación TP1
Luciano Fohrholtz	Dev Lead — arquitectura, clases del dominio, implementación de patrones
Juan Carlos Abente	QA Lead — patrón Strategy, AI_LOG, base para pruebas en TP2
Mariano Acosta	UX Lead — prototipo Figma, pantallas del caso de uso, normas ISO 9241-11

Sección 6

Uso de Inteligencia Artificial

El equipo utilizó herramientas de IA durante el desarrollo del TP1. Todo uso está registrado en el archivo docs/AI_LOG.md del repositorio, con detalle de herramienta, responsable, qué generó la IA, y qué modificaciones realizó el equipo.

6.1 Resumen de usos registrados

Entrada	Descripción del uso
001 — Semana 1	Identificación de proto-clases del sistema con sus atributos principales. Herramienta: Claude (Anthropic). Responsable: Dev Lead.
002 — Semana 1	Generación del diagrama de clases en formato Mermaid con tipos de relación. Herramienta: Claude. Responsable: Dev Lead.
003 — Semana 1	Selección y justificación de patrones de diseño GoF para el sistema. Herramienta: Claude. Responsable: Dev Lead.
004 — Semana 1	División de tareas del TP1 por integrante según rol y semana objetivo para el Kanban. Herramienta: Claude. Responsable: Scrum Master.

6.2 Criterio de uso crítico

En todos los casos, el equipo revisó el output generado por la IA antes de incorporarlo al proyecto. Las modificaciones realizadas incluyeron validación contra la consigna, ajuste de vocabulario al dominio real de la ferretería, y eliminación de elementos no pertinentes a la etapa del TP.

NOTA	Todos los integrantes del equipo comprenden el código y los patrones implementados, con independencia de si fueron asistidos por IA. El coloquio individual evaluará la comprensión técnica de cada uno.

Sección 7

Estado del Proyecto y Entregables

7.1 Checklist de entregables TP1

Entregable	Estado
Repositorio GitHub con estructura completa (docs/, src/, tests/, design/)	Completo
README con integrantes, roles, descripción y links	Completo
Tablero Kanban con 5 columnas y tarjetas del TP1	Completo
Matriz de riesgos (docs/matriz-de-riesgos.md)	Completo
Contrato de proyecto con metodología justificada	Completo
Clases del dominio implementadas en src/models.py	Completo
Patrón Observer implementado en el sistema	Completo
Patrón Strategy implementado en el sistema	Completo
Caso de uso principal funcionando punta a punta	Completo
Documentación docs/patrones-tp1.md	Completo
AI_LOG.md actualizado con todas las entradas	Completo
Prototipo Figma (design/)	Completo

7.2 Próximos pasos — TP2

- Implementar pruebas unitarias con TDD para las clases del dominio (QA Lead).
- Configurar automatización de pruebas con CI (QA Lead).
- Agregar nuevas estrategias de reporte (ReporteMovimientosDiarios) sin modificar código existente.
- Incorporar análisis estático con SonarCloud.
- Aplicar normas ISO 9241-11 e ISO 13407 en la validación de la interfaz (UX Lead).

Sección 8

Conclusión

El equipo implementó exitosamente los dos patrones de diseño requeridos por el TP1 — Observer y Strategy — integrándolos en funcionalidades reales del sistema de gestión de stock y no como ejemplos aislados.

Observer resuelve el desacoplamiento entre el estado de un producto y las acciones que se disparan cuando ese estado cambia. Strategy resuelve la variabilidad de algoritmos de reporte sin comprometer la estabilidad del código existente.

Ambos patrones fueron elegidos con justificación técnica basada en principios SOLID (SRP y OCP), pueden identificarse claramente en el código del repositorio, y sientan las bases para las extensiones que se desarrollarán en el TP2.

Referencias

- Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Consigna TP1 — Ingeniería de Software II · UCP · 2026.
- Guía de configuración del repositorio y tablero Kanban · IS2 UCP 2026.
- Rúbrica de coloquio individual · IS2 UCP 2026.

IS II · UCP Inc. · Informe TP1 · 2026