Proyecto Ciclo 4

Juan Nicolás Álvarez

Nicolás Sanchez

Escuela Colombiana de Ingeniería Julio Garavito

2025

**Restrospectiva:**

1**. ¿Cuáles fueron los mini-ciclos definidos (dentro del Ciclo 4)? Justifíquenlos.**

Mini-ciclo 1: Refactorización de la Herencia (La Base)

Tareas: Crear los dos paquetes (shapes, silkroad). Crear las clases abstractas Shape, Robot y Store. Modificar las clases existentes (como Circle, NormalRobot, NormalStore) para que heredaran de estas nuevas clases.

Justificación: Era el requisito principal de diseño. Necesitábamos reestructurar la base del proyecto y asegurar que el código antiguo siguiera funcionando bajo la nueva estructura de herencia, antes de añadir cualquier funcionalidad nueva.

Mini-ciclo 2: Implementación de Nuevos Tipos (La Extensión)

Tareas: Crear todas las nuevas clases hijas requeridas: AutonomousStore, FighterStore, NeverbackRobot y TenderRobot. Además, diseñar e implementar nuestro propio tipo: ThiefRobot.

Justificación: Cumplir con los nuevos requisitos funcionales . Este mini-ciclo se centró únicamente en crear las clases y su lógica interna (ej. FighterStore::collectTenges o NeverbackRobot::decideNextMove).

Mini-ciclo 3: Integración y Lógica Polimórfica (El Núcleo)

Tareas: Modificar la clase SilkRoad para que "entendiera" los nuevos tipos. Esto implicó crear los métodos sobrecargados placeStore(type, ...) y placeRobot(type, ...) y, lo más importante, actualizar la lógica central de moveRobot para que manejara polimórficamente las nuevas interacciones (el robo de ThiefRobot, la recolección de TenderRobot, etc.).

Justificación: Este era el corazón del ciclo. No bastaba con que las clases existieran, debían interactuar correctamente. La lógica de moveRobot tuvo que ser robustecida para manejar todos los nuevos casos especiales.

Mini-ciclo 4: Pruebas y Documentación de Diseño (La Validación)

Tareas: Crear los tres archivos de prueba requeridos (SilkRoadC4test, SilkRoadCC4test, SilkRoadAtest) . Actualizar todos los diagramas de Astah (crear el Diagrama de Paquetes, actualizar los Diagramas de Clases con la herencia y corregir los Diagramas de Secuencia).

Justificación: Cumplir con los requisitos de entregables. Necesitábamos probar formalmente que la nueva funcionalidad (Mini-ciclo 2) y la nueva lógica de integración (Mini-ciclo 3) funcionaban correctamente, además de documentar el nuevo diseño.

**2. ¿Cuál es el estado actual del proyecto en términos de mini-ciclos? ¿por qué?**

El proyecto está 100% completado.

Se han finalizado los 4 mini-ciclos definidos. El código de las 20 clases está implementado y compila correctamente. Todos los diagramas de Astah (paquetes, clases y secuencia) están terminados, corregidos y reflejan fielmente el código final. El proyecto cumple con todos los requisitos funcionales, de diseño y de usabilidad solicitados.

**3. ¿Cuál fue el tiempo total invertido por cada uno de ustedes? (Horas/Hombre)**

Juan Nicolás Álvarez: 10 horas

Nicolás Sánchez: 9 Horas

**4. ¿Cuál consideran fue el mayor logro? ¿Por qué?**

El mayor logro fue, sin duda, la refactorización exitosa del Ciclo 4.

Por qué: Logramos tomar un proyecto que ya era funcional (el del Ciclo 3) y aplicarle una reingeniería profunda basada en Herencia y Polimorfismo (creando las clases abstractas Robot y Store).

El éxito se mide en que pudimos añadir 5 nuevos tipos de elementos (Neverback, Tender, Thief, Autonomous, Fighter) modificando únicamente la lógica de creación en SilkRoad y sin "romper" el resto del simulador. El diagrama de clases final demuestra un diseño extensible y robusto, que era el objetivo final.

**5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?**

El mayor problema técnico fue asegurar la correcta interacción de las nuevas clases del Ciclo 4 durante el movimiento de los robots.

El método SilkRoad.moveRobot se volvió extremadamente complejo, ya que debía manejar múltiples lógicas:

Iterar por cada paso del camino.

Verificar si en un paso existía una tienda.

Llamar a store.collectTenges(robot), que a su vez tenía lógica polimórfica (la FighterStore podía rechazar al robot).

Verificar si el robot era TenderRobot para dividir la ganancia.

Verificar si el robot era ThiefRobot para intentar robar a otro robot en la misma casilla.

Solución: La clave fue usar una combinación de diagramas de secuencia detallados y pruebas de aceptación. Corregimos el sd moveRobot varias veces para modelar todos los fragmentos loop, opt y alt (para ThiefRobot y TenderRobot). Luego, las pruebas de aceptación (SilkRoadAtest) nos permitieron verificar visualmente que esa lógica compleja se estaba ejecutando como esperábamos.

**6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?**

Qué se hizo bien: Tuvimos una excelente comunicación iterativa. No intentamos resolver todo de una vez. Seguimos un flujo constante de: Implementar una funcionalidad (ej. crear el ThiefRobot).

Modelarla en Astah (crear el diagrama de clases y secuencia).

Identificar inconsistencias entre el código y el diagrama.

Corregir el diagrama o el código hasta que ambos estuvieran alineados. Este enfoque nos permitió detectar errores de lógica (como los de sd moveRobots o sd reboot) antes de que se volvieran problemas mayores.

Compromiso para mejorar: Establecer estándares de codificación y nomenclatura desde el inicio.

**7. Considerando las prácticas XP incluidas en los laboratorios. ¿cuál fue la más útil? ¿por qué?**

La práctica más útil fue, sin duda, el Desarrollo Guiado por Pruebas (TDD).

Por qué: El proyecto estaba estructurado para forzar esta práctica. Cada ciclo venía con un conjunto de pruebas de unidad y comunes (SilkRoadCC4test).

Fue imposible implementar la refactorización del Ciclo 4 "a ciegas". La única forma de asegurarnos de que al crear la clase abstracta Robot no habíamos dañado el método moveRobots, era correr las pruebas del Ciclo 2. Las pruebas se convirtieron en una "red de seguridad" que nos dio la confianza para hacer cambios profundos en la estructura del código. Las pruebas de aceptación (SilkRoadAtest) fueron igual de cruciales para validar la lógica visual que las pruebas de unidad no podían capturar.

**8. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.**

Documentación de Java: Se utilizó la documentación oficial de Oracle para consultar el funcionamiento de colecciones y componentes de Swing.

Oracle. (2025). Java® Platform, Standard Edition 21 API Specification. Recuperado de https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Tutoriales de Astah: Se consultaron guías de Astah para el modelado de diagramas de secuencia complejos.

Astah. (2025). Astah UML User Guide - Sequence Diagram. Recuperado de https://astah.net/guide/uml/user-guide/sequence\_diagram.html