



Guía de Evaluación: Semana 1 - Ingeniería y Calidad

Taller: Refactorización AI-Native y Arquitecturas Hexagonales

1. Objetivo de la Evaluación

Validar que el estudiante es capaz de trascender el "código que funciona" para construir "código profesional". Se evaluará la transición de un MVP acoplado hacia una arquitectura desacoplada, testeable y mantenible, utilizando la IA como un auditor crítico y no solo como un generador de código.

2. El Reto: "La Auditoría de Acoplamiento"

Los equipos deben demostrar que su lógica de negocio (Dominio) ha quedado blindada contra cambios en la infraestructura. La prueba definitiva será la ejecución de **Tests Unitarios en aislamiento total** y la sustentación del plan de ataque contra la deuda técnica inicial.

3. Conceptos Clave a Aplicar (Temas de Investigación)

Para aprobar, el proyecto debe evidenciar la aplicación práctica de:

1. **Principios SOLID (Acrónimo Completo):** Aplicación de los 5 pilares para reducir el acoplamiento y aumentar la cohesión.
2. **Arquitectura Hexagonal:** Separación clara por Puertos y Adaptadores, aislando el Núcleo (Dominio y Aplicación) de la Infraestructura.
3. **Patrones de Diseño por Categorías:** Los estudiantes deben **usar, aplicar y explicar** la decisión técnica tras elegir patrones:
 - **Creacionales:** Factory, Singleton, etc.
 - **Estructurales:** Repository, Adapter, Proxy, etc.
 - **Comportamiento:** Strategy, Command, Observer, etc.
4. **Unit Testing vs. Integration Testing:** Pruebas que usan Mocks/Stubs para evitar dependencias externas.
5. **AI-Native Auditing:** Uso de la IA para auditar SOLID y proponer refactorizaciones.

4. Entregables Obligatorios

- **DEBT_REPORT.md:** Informe de los "Pecados Capitales" iniciales y su mitigación con colaboración Humano-IA.
- **Código Refactorizado:** Repositorio con Arquitectura Hexagonal.
- **Suite de Tests:** Flujo crítico probado sin dependencias externas.

5. Rúbrica de Evaluación (Semana 1)

Criterio	Nivel Experto (5.0)	Nivel Competente (3.5)	Nivel Insuficiente (2.0)
Arquitectura Hexagonal	Separación absoluta. El dominio/aplicación no importan librerías de infraestructura. Puertos bien definidos.	Existe separación, pero hay lógica de negocio filtrada en los adaptadores o viceversa.	Lógica de negocio mezclada con controladores, ORMs o clientes de mensajería.
Principios SOLID	Se evidencia la aplicación del acrónimo completo. Código altamente cohesivo y desacoplado.	Se aplican algunos principios (ej. SRP, DIP) pero se violan otros (ej. ISP o OCP).	El código ignora los principios SOLID; clases gigantes y acoplamiento rígido.
Patrones de Diseño	Uso correcto de patrones en múltiples categorías. Justifica técnicamente la elección de cada uno.	Implementa patrones estructurales (ej. Repository) pero falla en usarlos o explicarlos bien.	No se identifican patrones de diseño o se usan de forma incorrecta (Anti-patrones).
Testing y Aislamiento	Tests unitarios puros que usan Mocks para los puertos de salida. Cobertura lógica total.	Hay tests unitarios, pero dependen parcialmente de configuraciones de entorno o DB.	Solo existen tests de integración o el código es "imposible de testear" por su acoplamiento.
Sustentación y Human Check	Defensa técnica impecable. Código legible y balanceado con IA.	Explica el código pero tiene vacíos conceptuales sobre por qué usó un patrón específico.	No puede explicar decisiones de diseño; el código parece una caja negra generada

			por IA.
--	--	--	---------

6. Formato de Presentación (Viernes PM)

1. **El Diagnóstico:** Muestra de los "Pecados Capitales" originales en el DEBT_REPORT.md.
2. **La Solución:** Demo del código refactorizado y ejecución de la suite de tests en aislamiento.
3. **Defensa:** Justificación de los patrones de diseño elegidos por categorías.