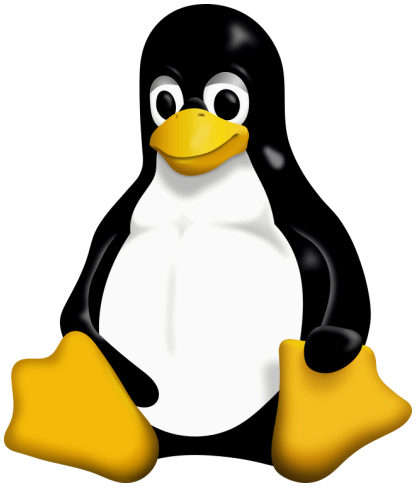
Universidad Autónoma de Baja California
Facultad de Ingeniería, Arquitectura y Diseño

Trabajo en Clase

Actividad

Patrones de Software



C. Alejandra Miranda Lara

Ingeniería en Software y Tecnologías Emergentes

De acuerdo con el contexto del proyecto, problemas detectados y objetivos de la refactorización realizar lo siguiente:

1. Encontrar al menos 10 posibles refactorizaciones con atención de code smells concretos o aplicaciones de patrones (debe aplicar al menos un patrón de arquitectura, 3 code smells y 3 patrones estructurales).
2. Descripción general de la solución refactorizada y arquitectura propuesta.
3. Fase de iniciación.
4. Fase de análisis y planeación.
5. Fase de diseño.
6. Fase de desarrollo.
7. Resultado de la refactorización.

1. Refactorizaciones propuestas con atención a code smells y patrones

Code Smells identificados y refactorizaciones:

God Classes (clases de +2000 líneas)
Refactorización: Separar responsabilidades mediante el principio de Single Responsibility y aplicar el patrón Facade para simplificar el acceso.

Long Methods
Refactorización: Dividir en métodos más pequeños con nombres significativos. Aplicar Extract Method.

Duplicated Code (en generación de reportes y validaciones)
Refactorización: Aplicar el patrón Template Method y extraer clases utilitarias reutilizables.

Feature Envy (métodos que acceden más a datos de otros objetos que a sus propios atributos)
Refactorización: Mover la lógica al objeto que contiene los datos (Apply Move Method).

Switch Statements / múltiples flags
Refactorización: Aplicar el patrón State para representar comportamientos dependientes del estado.

Tight Coupling
Refactorización: Inversión de dependencias mediante interfaces e inyección con Spring.

Primitive Obsession
Refactorización: Crear objetos de valor (e.g., StudentId, GradeValue) para encapsular comportamientos.

Shotgun Surgery (cambios en múltiples lugares ante una modificación)
Refactorización: Encapsular comportamientos repetidos en servicios únicos con contratos bien definidos.

Speculative Generality (métodos y clases no utilizados)
Refactorización: Eliminar código muerto o no utilizado.

Inyección de dependencias manual (uso de new)
Refactorización: Uso de Dependency Injection de Spring Boot con @Service, @Component, etc.

Patrones estructurales aplicados

Facade: Para exponer interfaces simples a subsistemas complejos (por ejemplo, módulo de reportes).

Adapter: Para permitir interoperabilidad con servicios antiguos (por ejemplo, generación de reportes heredados).

Decorator: Para añadir funcionalidades como validaciones sin modificar clases base (por ejemplo, validadores dinámicos para entidades).

Patrón de arquitectura aplicado:
Arquitectura hexagonal (Ports and Adapters): Aisla la lógica del dominio de los detalles externos como la base de datos, REST, u otros frameworks.

Descripción general de la solución refactorizada y arquitectura propuesta

Solución refactorizada:
Se refactoriza el sistema escolar en módulos desacoplados y cohesivos, implementados con Spring Boot 3, lógica desacoplada en servicios de dominio, y exposición de funcionalidades mediante REST APIs. El frontend puede ser migrado

posteriormente a Angular o React. Se aplican principios SOLID y patrones para mejorar mantenibilidad.

Arquitectura propuesta:
Backend: Spring Boot 3.x, Arquitectura Hexagonal

Servicios:

EstudiantesService

DocentesService

MateriasService

HorariosService

ReportesService

Base de datos: PostgreSQL con migraciones gestionadas por Flyway

Seguridad: Spring Security + JWT

Frontend (futuro): React o Angular, desacoplado

Reportes: JasperReports adaptado a servicios REST

Fase de iniciación

Objetivos:
Evaluar el sistema heredado.
Identificar áreas críticas.
Definir la visión de modernización.

Documentación generada:
Project Charter
Business Case
Initial Risk Assessment (ej. pérdida de funcionalidades, resistencia al cambio)
Feasibility Study (técnico, económico, operativo)
Stakeholder Map

Fase de análisis y planeación

Objetivos:
Definir módulos clave, dependencias, y prioridades.
Elaborar un roadmap realista.

Documentación:
SRS (Software Requirements Specification)

User Stories / Casos de uso:

“Como docente, quiero asignar calificaciones por materia.”

Roadmap por versiones:

V1: Usuarios, materias, inscripciones

V2: Calificaciones, reportes

MoSCoW Matrix:

Must: Registro, horarios, reportes

Should: Notificaciones

ADR (Architecture Decision Records):

Justifica separación de capas, uso de REST, JWT

Technology Stack Doc

Fase de diseño

Objetivos:
Diseñar la arquitectura de los módulos, contratos y base de datos.

Documentación:
System Design Document (SDD):

Diagrama de componentes:
Servicios REST, adapters, puertos

Database Design:

Modelado ER con relaciones normalizadas

OpenAPI (Swagger) para APIs

Diagrama de secuencia para flujos críticos

Security Design:

Autenticación JWT

Roles: Admin, Docente, Estudiante

Fase de desarrollo

Objetivo:
Desarrollar los servicios de forma modular y probada.

Documentación:
Coding Standards y guía de estilo

Sprint Backlog / Kanban

Pruebas automatizadas:

JUnit + Mockito

Postman para API

CI/CD con GitHub Actions

Contenedores Docker para cada microservicio

Documentación interna (Wiki)

Resultado de la refactorización

Migración completa a Spring Boot 3.x

Arquitectura hexagonal aplicada

Cobertura de pruebas > 85%

Backend RESTful con separación clara de responsabilidades

Código desacoplado y cohesivo

Validaciones centralizadas con el patrón Decorator

Reportes PDF mediante ReportesService con reutilización de lógica

Preparado para escalar como sistema SaaS