Codelab:

Proceso con enfoque ADD y Clean Architecture

Desarrollo de Software III

Miguel Angel Ceballos Yate – 2259515

Docente:

Salazar, ÁLVARO

Sede Tuluá

Junio de 2025

1. ¿Qué es Attribute-Driven Design (ADD) y cuál es su propósito en el diseño de software?

Attribute-Driven Design (ADD) es una técnica de diseño de arquitectura de software que se enfoca en los **atributos de calidad** como rendimiento, seguridad o disponibilidad para guiar las decisiones arquitectónicas. Su propósito es asegurar que la arquitectura del sistema no solo cumpla con los requerimientos funcionales, sino también con los atributos de calidad definidos por el negocio, descomponiendo el sistema en módulos y eligiendo tácticas de diseño que garanticen dichas cualidades.

2. ¿Cómo se relaciona ADD con Clean Architecture en el proceso de diseño de sistemas?

ADD (Attribute-Driven Design) y **Clean Architecture** se relacionan en que ambos buscan crear sistemas bien estructurados y sostenibles, pero lo hacen desde enfoques complementarios:

- **ADD** se enfoca en los **atributos de calidad** (como rendimiento, seguridad o escalabilidad) como guía principal para tomar decisiones arquitectónicas durante el diseño. Es un proceso **orientado a requerimientos no funcionales**.

- **Clean Architecture**, en cambio, se centra en separar claramente las responsabilidades del sistema en capas (entidades, casos de uso, interfaz, etc.), promoviendo la independencia del código frente a detalles externos como bases de datos o interfaces gráficas.

Relación entre ambos:

En un proceso de diseño completo:

- **ADD puede usarse primero** para identificar los atributos críticos del sistema y tomar decisiones clave (por ejemplo, uso de caché para rendimiento o replicación para disponibilidad).

- Luego, **Clean Architecture puede aplicarse como marco estructural** para organizar esas decisiones en una arquitectura clara, desacoplada y mantenible.

3. ¿Cuáles son los pasos principales del método ADD para definir una arquitectura de software?

Los **pasos principales del método ADD (Attribute-Driven Design)** para definir una arquitectura de software son los siguientes:

1. **Identificar los requerimientos del sistema**
   Incluye funcionalidades, restricciones y especialmente los **atributos de calidad** (como rendimiento, seguridad, disponibilidad, etc.).

2. **Seleccionar el módulo a diseñar**
   Se empieza por el sistema completo o un módulo importante y se diseña desde arriba hacia abajo.

3. **Seleccionar los escenarios de calidad relevantes**

Se eligen los atributos de calidad más críticos que deben cumplirse en ese módulo.

4. **Elegir tácticas de diseño apropiadas**
   Se aplican soluciones arquitectónicas concretas para cumplir los atributos de calidad (por ejemplo, usar caché para mejorar rendimiento).

5. **Descomponer el módulo en subcomponentes**
   Se divide en partes más pequeñas con responsabilidades claras.

6. **Asignar responsabilidades y definir interfaces**
   Se definen qué hace cada componente y cómo se comunican entre ellos.

7. **Registrar decisiones arquitectónicas**
   Se documentan las decisiones tomadas y su justificación.

8. **Repetir el proceso para cada subcomponente**
   Se aplica recursivamente hasta llegar al nivel de detalle deseado.

4. ¿Cómo se identifican los atributos de calidad en ADD y por qué son importantes?

En ADD, los **atributos de calidad** se descubren al inicio del proyecto a través de una **recopilación estructurada de expectativas de los interesados** (clientes, usuarios finales, equipo de negocio, operaciones, etc.). El arquitecto suele (1) revisar la especificación de requisitos y restricciones, (2) conducir entrevistas y talleres colaborativos, y (3) construir un *utility tree* o lista priorizada de escenarios ATAM, donde cada atributo se traduce en uno o más **escenarios medibles** (⟨*estímulo*, *entorno*, *respuesta esperada*, *métrica*⟩). Estos escenarios se ponderan según el impacto de negocio y el riesgo técnico para destacar los "atributos de calidad clave" que realmente deben guiar el diseño.

Son importantes porque:

- **Dirigen todas las decisiones arquitectónicas**: las tácticas, patrones y particiones del sistema se eligen específicamente para satisfacer los escenarios priorizados.

- **Hacen visible el riesgo temprano**: obligan a discutir trade‑offs (p. ej., rendimiento vs. seguridad) antes de escribir código.

- **Proveen criterios de verificación**: al estar especificados como escenarios medibles, permiten evaluar la arquitectura con pruebas, prototipos o revisiones formales.

- **Alinean tecnología y negocio**: garantizan que la arquitectura responda a lo que realmente importa para el éxito del sistema (por ejemplo, latencia de 200 ms para retención de usuarios o cumplimiento de una norma de seguridad).

En suma, identificar bien los atributos de calidad es el paso crítico que transforma requisitos abstractos en criterios concretos y comprobables que gobiernan todo el proceso de diseño en ADD.

5. ¿Por qué Clean Architecture complementa ADD en la implementación de una solución?

Clean Architecture complementa ADD porque, mientras ADD se enfoca en tomar decisiones arquitectónicas basadas en los atributos de calidad del sistema (como rendimiento, seguridad o disponibilidad), Clean Architecture ofrece una estructura clara y desacoplada para implementar esas decisiones. Es decir, ADD define **qué debe lograrse** en términos de calidad y comportamiento del sistema, y Clean Architecture define **cómo organizar el código** para lograrlo de forma mantenible, modular y flexible. Así, juntos permiten diseñar e implementar soluciones que no solo cumplen los requisitos técnicos, sino que también son fáciles de entender, probar y evolucionar.

6. ¿Qué criterios se deben considerar al definir las capas en Clean Architecture dentro de un proceso ADD?

- Separación de responsabilidades.
- Independencia de tecnologías y frameworks.
- Inversión de dependencias (el dominio no depende de detalles externos).
- Reutilización y testabilidad del código.
- Soporte para atributos de calidad como rendimiento (mediante capas optimizadas) o seguridad (control de acceso en la capa adecuada).

7. ¿Cómo ADD ayuda a tomar decisiones arquitectónicas basadas en necesidades del negocio?

ADD traduce las necesidades del negocio en atributos de calidad medibles. A partir de estos, el arquitecto selecciona tácticas adecuadas (ej. usar caché para rendimiento, encriptación para seguridad), asegurando que cada decisión técnica tenga un respaldo en los objetivos estratégicos del sistema.

8. ¿Cuáles son los beneficios de combinar ADD con Clean Architecture en un sistema basado en microservicios?

- Diseño enfocado en calidad desde el inicio.
- Servicios desacoplados, orientados a casos de uso.
- Alta mantenibilidad y facilidad de prueba.
- Capacidad de adaptar o escalar servicios sin romper la lógica del dominio.
- Facilidad para validar que cada microservicio cumple sus atributos clave (como disponibilidad o rendimiento).

9. ¿Cómo se asegura que la arquitectura resultante cumpla con los atributos de calidad definidos en ADD?

Se realiza una fase de **validación y refinamiento**, que incluye:

- Revisiones arquitectónicas.
- Pruebas de rendimiento, carga y seguridad.
- Análisis de cuellos de botella.
- Ajustes de implementación o diseño (ej. agregar caching, mejorar consultas, replicar servicios).

10. ¿Qué herramientas o metodologías pueden ayudar a validar una arquitectura diseñada con ADD y Clean Architecture?

- Revisiones por pares o arquitectos (ARB - Architecture Review Board).
- Pruebas de carga y estrés (ej. JMeter, Gatling).
- Pruebas de seguridad (ej. OWASP ZAP, SonarQube).
- Análisis estático de código (ej. SonarQube, PMD).
- Monitoreo y observabilidad (ej. Prometheus + Grafana).
- Modelos de evaluación de arquitectura como ATAM (Architecture Tradeoff Analysis Method).