ADD + CLEAN ARCHITECTURE CODELAB
2259519 - Juan Pablo Escobar

Preguntas:

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

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

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

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

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

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

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

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

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

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

Solución:

a. **ADD (Attribute-Driven Design)** es un enfoque sistemático para el diseño arquitectónico de software que da prioridad a los atributos de calidad (requisitos no funcionales) sobre los funcionales.

   Su finalidad es: Asegurar que la arquitectura cumpla con las necesidades críticas de calidad como el rendimiento, la seguridad o la disponibilidad en sus inicios, además de tomar decisiones basadas en las tácticas de la arquitectura para cada atributo. Busca, además, el limitar el riesgo técnico al enfrentarse a un problema (por ejemplo: escalabilidad en un sistema distribuido). Y busca evitar soluciones generales y tener un diseño orientado hacia el tipo de negocio que se desea soportar.

b. **ADD y Clean Architecture** tienen una relación de complementariedad, donde ADD determina qué atributos de calidad debe cumplir el sistema en el que se aplicaría (un ejemplo es que debe tener una alta escalabilidad), mientras que Clean Architecture define cómo implementar el diseño, a través de una estructura de código que no tiene interdependencias (es decir que está desacoplado). En este sentido, ADD es la guía de las decisiones de diseño a nivel macro (componentes, interacciones), y Clean Architecture hace esta traducción en capas (Dominio, Aplicación) que aseguran su independencia respecto a tecnologías externas. Aplicándolo a un ejemplo, si de ADD se deriva la necesidad de seguridad, la Clean Architecture la implementará a través del cifrado en la capa de Infraestructura, independientemente de la lógica de negocio.
c. Los pasos clave de **ADD** son:
   Identificar requisitos: Atributos de calidad prioritarios (ej: rendimiento < 2 seg) y restricciones (ej: uso de microservicios).
   Definir escenarios arquitectónicos: Cómo se puede ver cada atributo (ej: "soportar 10K solicitudes concurrentes").
   Seleccionar tácticas: Solución técnica para cada atributo (ej: caching para rendimiento).
   Diseñar estructura: Definir módulos (ej: Servicio de Autenticación) y sus interacciones
   Validar y refinar: Verificar cómo se cumple el atributo a través de pruebas (ej: pruebas de carga) y mejorar la arquitectura.

– Mejorar respuesta:

d. Los atributos de calidad se identifican mediante cosas como:

Requisitos del negocio (Ejemplo de este: "disponibilidad 99.9%" para un sistema bancario).

Regulaciones (Un ejemplo sería: GDPR para seguridad de datos).

Expectativas de usuarios (Un ejemplo sería: respuesta en menos de 1 segundo).

La importancia de estos aspectos radica en que son clave para determinar si el sistema es técnicamente viable. Si hay fallos en atributos como la seguridad o el rendimiento, implica inmediatamente un fracaso funcional, incluso si las características operativas están en orden. Además, estos factores afectan los costos (por ejemplo, una alta disponibilidad requiere una infraestructura redundante) y establecen las prioridades estratégicas (como la baja latencia en sistemas de trading).

e. Clean Architecture complementa ADD al traducir los atributos de calidad definidos por ADD en una estructura de código mantenible y desacoplada. Mientras ADD establece requisitos no funcionales, Clean Architecture garantiza su implementación mediante inversión de dependencias, separación de capas y flexibilidad.

f. Los criterios clave son:

Atributos de calidad prioritarios: Si ADD exige seguridad, la capa de Infraestructura debe implementar cifrado; si prioriza rendimiento, se agrega caching en la misma capa.

Desacoplamiento: Las capas deben aislar responsabilidades (ej: Dominio puro, sin dependencias de frameworks).

Escenarios de cambio: Anticipar áreas con alta probabilidad de modificación (Como las bases de datos o interfaces de usuario), ubicándose en capas externas (Presentación/Infraestructura).

Complejidad del negocio: Si la lógica es crítica (transacciones), la capa de Dominio debe ser robusta y evaluable independientemente.

g. ADD vincula necesidades del negocio con soluciones técnicas tal que:

   Priorización de atributos: Convierte objetivos comerciales en atributos medibles. (Un ejemplo sería transformar “Alto rendimiento y velocidad de carga” en mejoras en rendimiento para estar por debajo de los 2 segundos de carga).

   Selección de tácticas alineadas: ADD elige tácticas como autoescalado en la nube, reduciendo costos operativos.

   Mitigación de riesgos: Si el negocio requiere continuidad, ADD impone tácticas de alta disponibilidad (realiza réplicas en múltiples regiones).

   Validación temprana: Las pruebas de concepto evalúan si las decisiones tomadas soportan las metas del negocio antes de implementarlas.

h. Esta combinación ofrece:

   Definición clara de responsabilidades: ADD asigna atributos de calidad por microservicio. Clean Architecture asegura que cada microservicio mantenga capas internas desacopladas.

   Escalabilidad granular: Las tácticas de ADD se implementan vía Clean Architecture, permitiendo escalar servicios críticos sin afectar otros.

   Resistencia y autonomía: Fallos en un microservicio no propagan errores, dónde ADD proporciona tolerancia a fallos mientras Clean Architecture aísla componentes.

   Evolución independiente: Los cambios en un servicio no impactan el núcleo de negocio gracias a las capas de Clean Architecture.

i. Se validan mediante pruebas técnicas específicas (De rendimiento, seguridad, disponibilidad, entre otras.), métricas cuantitativas (Como monitoreo continuo para mantener latencias bajas, por ejemplo), revisión de la arquitectura (buscando evaluar el diseño frente a escenarios especificados, como un alto flujo de usuarios) o las iteraciones (buscando refinar tácticas en caso tal de que se incumplan atributos).

j. Herramientas clave:

Pruebas: JMeter (rendimiento), OWASP ZAP (seguridad), SonarQube (calidad de código).

Monitoreo: Prometheus/Grafana (métricas en tiempo real), ELK Stack (logs).

Modelado: Diagrams.net o Lucidchart para visualizar componentes y flujos.

Metodologías: ATAM (Architecture Tradeoff Analysis Method) para priorizar atributos, y TDD/BDD para garantizar que la implementación en capas cumple los requisitos.