**1. ¿Qué es Attribute-Driven Design (ADD)?**

ADD es un método sistemático para diseñar arquitecturas de software que se centra en los atributos de calidad como criterio principal de diseño. Su propósito es crear arquitecturas que satisfagan explícitamente los requisitos no funcionales (rendimiento, seguridad, mantenibilidad, escalabilidad) desde las primeras etapas del diseño.

**2. Relación entre ADD y Clean Architecture**

ADD y Clean Architecture son complementarios:

-ADD proporciona el proceso para tomar decisiones arquitectónicas basadas en atributos de calidad

-Clean Architecture ofrece un patrón estructural que naturalmente satisface muchos atributos de calidad (mantenibilidad, testabilidad, independencia de frameworks)

-ADD puede usar Clean Architecture como una de sus tácticas arquitectónicas para lograr separación de responsabilidades y reducir acoplamiento.

**3. Pasos principales del método ADD**

El proceso ADD sigue estos pasos iterativos:

Paso 1: Confirmar los drivers arquitectónicos (requisitos funcionales, atributos de calidad, restricciones)

Paso 2: Elegir el elemento del sistema a descomponer (inicialmente todo el sistema)

Paso 3: Identificar candidatos de diseño arquitectónico (patrones, tácticas, referencias)

Paso 4: Seleccionar uno o más elementos de diseño basándose en los drivers

Paso 5: Instanciar elementos arquitectónicos y asignar responsabilidades

Paso 6: Definir interfaces entre elementos

Paso 7: Verificar y refinar requisitos, hacer elementos candidatos para iteración

**4. Identificación de atributos de calidad en ADD**

Los atributos de calidad se identifican mediante:

Técnicas de elicitación:

- Entrevistas con stakeholders

-Análisis de escenarios de calidad

-Workshops de arquitectura

-Revisión de sistemas similares

Importancia:

-Guían las decisiones arquitectónicas

-Permiten evaluar trade-offs

-Establecen criterios de éxito medibles

-Priorizan inversiones técnicas

Los atributos típicos incluyen: performance, availability, security, modifiability, usability, testability.

**5. Por qué Clean Architecture complementa ADD**

Clean Architecture complementa ADD porque:

Satisface múltiples atributos inherentemente:

- Mantenibilidad: Separación clara de responsabilidades

- Testabilidad: Inversión de dependencias facilita mocking

- Flexibilidad: Cambios en capas externas no afectan lógica de negocio

-Independencia: Desacoplamiento de frameworks, UI, bases de datos

Proporciona estructura predecible que ADD puede aplicar consistentemente en diferentes contextos.

**6. Criterios para definir capas en Clean Architecture con ADD**

Al aplicar ADD para definir capas, considera:

Criterios de separación:

-Responsabilidad: Cada capa tiene un propósito específico

- Volatilidad: Elementos que cambian juntos van en la misma capa

- Dependencias: Flujo unidireccional hacia el centro

-Abstracción: Niveles crecientes hacia el núcleo

Atributos de calidad específicos:

- Security: Validaciones en capas apropiadas

- Performance: Optimizaciones sin violar principios

- Scalability: Separación que permita escalado independiente

**7. Decisiones arquitectónicas basadas en necesidades del negocio**

ADD conecta negocio con arquitectura mediante:

Traducción de requisitos: Convierte necesidades de negocio en escenarios de atributos de calidad medibles

Priorización: Usa criterios de negocio para priorizar atributos cuando hay conflictos

Justificación de decisiones: Cada elemento arquitectónico se relaciona directamente con objetivos de negocio

Trazabilidad: Mantiene vínculos claros entre decisiones técnicas y requerimientos de negocio

**8. Beneficios en sistemas de microservicios**

La combinación ADD + Clean Architecture en microservicios ofrece:

A nivel de servicio individual:

-Cada microservicio sigue Clean Architecture internamente

- Facilita testing y mantenimiento independiente

A nivel de sistema:

-ADD ayuda a definir boundaries entre servicios

- Atributos de calidad guían decisiones de particionado

- Consistencia arquitectónica entre servicios

Beneficios específicos:

- Scalability: Servicios independientes escalables

- Reliability: Fallas aisladas

- Deployability: Despliegues independientes

**9. Asegurar cumplimiento de atributos de calidad**

Para validar el cumplimiento:

**Durante el diseño:**

* Revisiones arquitectónicas por escenarios
* Análisis de trade-offs (ATAM)
* Prototipos arquitectónicos

**Durante implementación:**

* Métricas de calidad automatizadas
* Testing de atributos no funcionales
* Monitoreo continuo

**Técnicas específicas:**

* **Performance**: Load testing, profiling
* **Security**: Penetration testing, code analysis
* **Maintainability**: Métricas de código, revisiones

**10. Herramientas y metodologías de validación**

**Métodos de evaluación:**

* **ATAM** (Architecture Tradeoff Analysis Method)
* **SAAM** (Software Architecture Analysis Method)
* **ARID** (Active Reviews for Intermediate Designs)

**Herramientas técnicas:**

* **Análisis estático**: SonarQube, NDepend
* **Testing de performance**: JMeter, LoadRunner
* **Monitoreo**: Application Performance Monitoring (APM)
* **Documentación**: Architecture Decision Records (ADRs)

**Validación continua:**

* **Fitness functions**: Tests automatizados de características arquitectónicas
* **Chaos engineering**: Validación de resilencia
* **A/B testing**: Validación de atributos en producción

La clave está en establecer un proceso continuo de medición y retroalimentación que permita ajustar la arquitectura conforme evolucionan las necesidades del negocio y se obtiene nueva información sobre el comportamiento del sistema.