Universidad de Oviedo

ES
Español


# Alcanzando la arquitectura del software


Curso 2020/2021

Jose Emilio Labra Gayo

# Alcanzando la arquitectura del software

- Conceptos de diseño
  - Tácticas, estilos, patrones, arquitecturas de referencia, componentes externos
- Metodologías
  - ADD
- Toma de decisiones
- Incidencias arquitectónicas
- Evaluación de arquitecturas

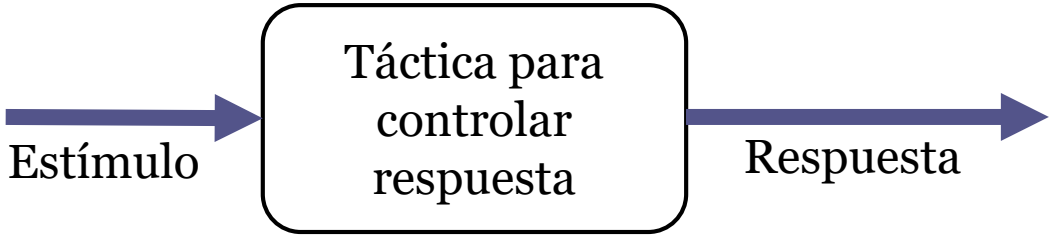
# Tácticas

Técnicas de diseño para alcanzar una respuesta a algunos atributos de calidad

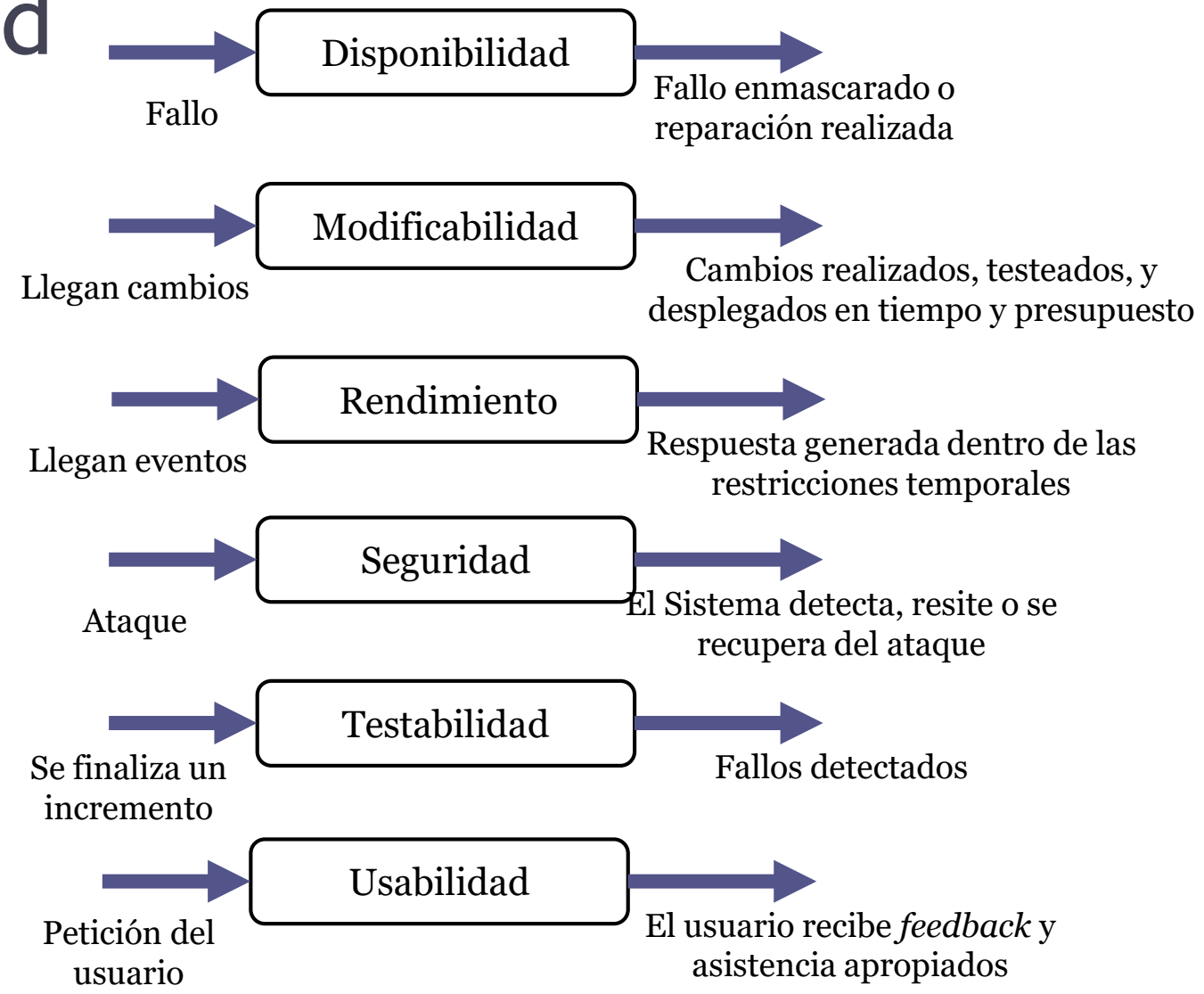
Las tácticas se enfocan en la respuesta a un Atributo de calidad

Pueden chocar con otros atributos de calidad

Las tácticas intentan controlar respuestas a estímulos

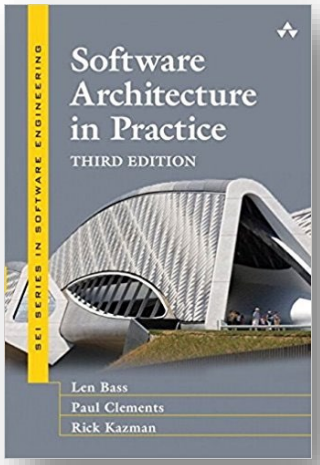


# Tácticas dependen del atributo de calidad



# ¿Dónde podemos encontrar tácticas?

- Propia experiencia del arquitecto
- Experiencia documentada de la comunidad
  - Libros, conferencias, blogs,...
- Las tácticas evolucionan con tiempo y tendencias
  - Libro "Software architecture in practice" contiene una lista de varias tácticas

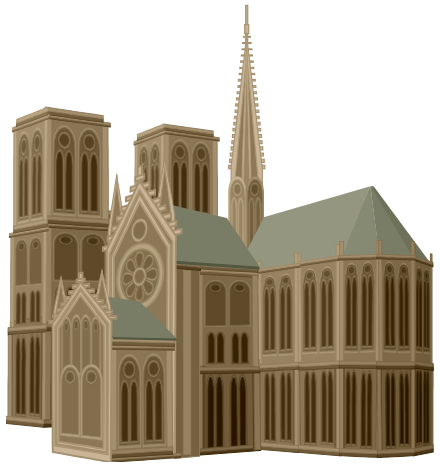



http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05lev1sec1.html
https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm

# Estilos arquitectónicos

Definen la forma general del sistema

- Contienen:
  - Elementos: Componentes que contienen funcionalidad
  - Relaciones: Relaciones entre los elementos
  - Restricciones: Limitan la integración entre elementos
  - Lista de atributos:
    - Ventajas/desventajas de un estilo



# ¿Existen estilos puros?

Estilos puros = idealización

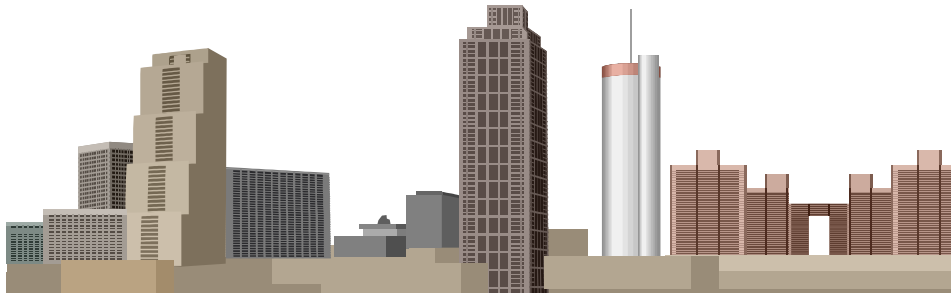
En práctica, los estilos puros se dan pocas veces

Normalmente, los sistemas se desvían de estilos puros...

...o combinan varios estilos arquitectónicos

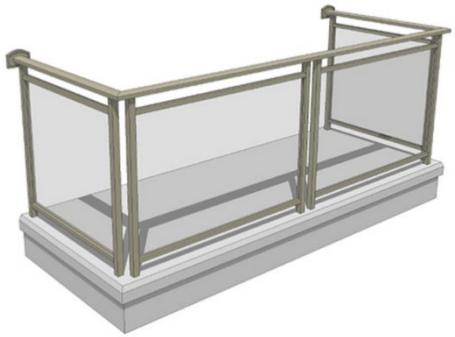
Es importante comprender los estilos puros para:

- Comprender pros/cons de un estilo
- Valorar las consecuencias de desviarse del estilo



# Patrón arquitectónico

- Solución general y reutilizable a algún problema recurrente que aparece en un contexto
  - Parámetro importante: **problema**
- 3 tipos:
  - Estructurales: Tiempo de construcción
    - Ejemplo: Layers
  - Runtime (comportamiento)
    - Ejemplo: Pipes & filters
  - Despliegue
    - Ejemplo: Cluster de balanceo de carga



# Patrón vs Estilo

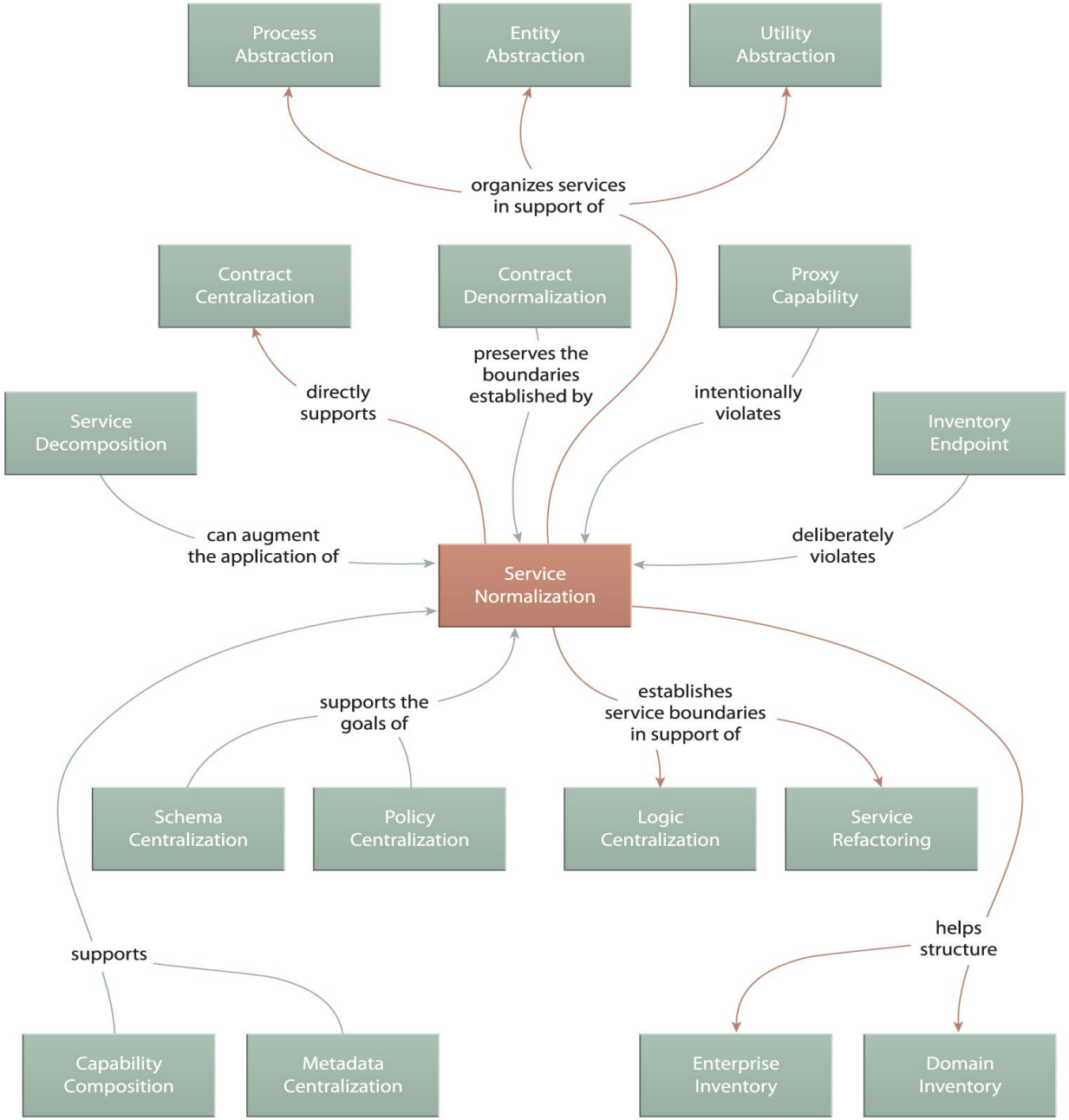
- Patrón = solución a un problema
  - Estilo = genérico
    - No tiene que estar asociado a un problema
- Estilo define la arquitectura general de una aplicación
  - Normalmente, una aplicación tiene un estilo
    - ...pero puede tener varios patrones
  - Los patrones aparecen en escalas diferentes
    - Alto nivel (patrones arquitectónicos)
    - Diseño (patrones de diseño)
    - Implementación (idiomas)
    - . . .

# Patrón vs Estilo

- Los estilos, en general, son independientes entre sí
- Un patron puede relacionarse con otros patrones
  - Un patrón puede estar compuesto de varios patrones
  - Pueden crearse interacciones entre patrones

# Lenguajes y catálogos de patrones

- Catálogo de patrones
  - Un conjunto de patrones sobre un asunto
    - No tiene porqué ser exhaustivo
- Lenguaje de patrones
  - Un catálogo de patrones completo sobre un tema
    - Objetivo: documentar todas las posibilidades
    - Normalmente incluyen relaciones entre patrones
      - Mapa gráfico



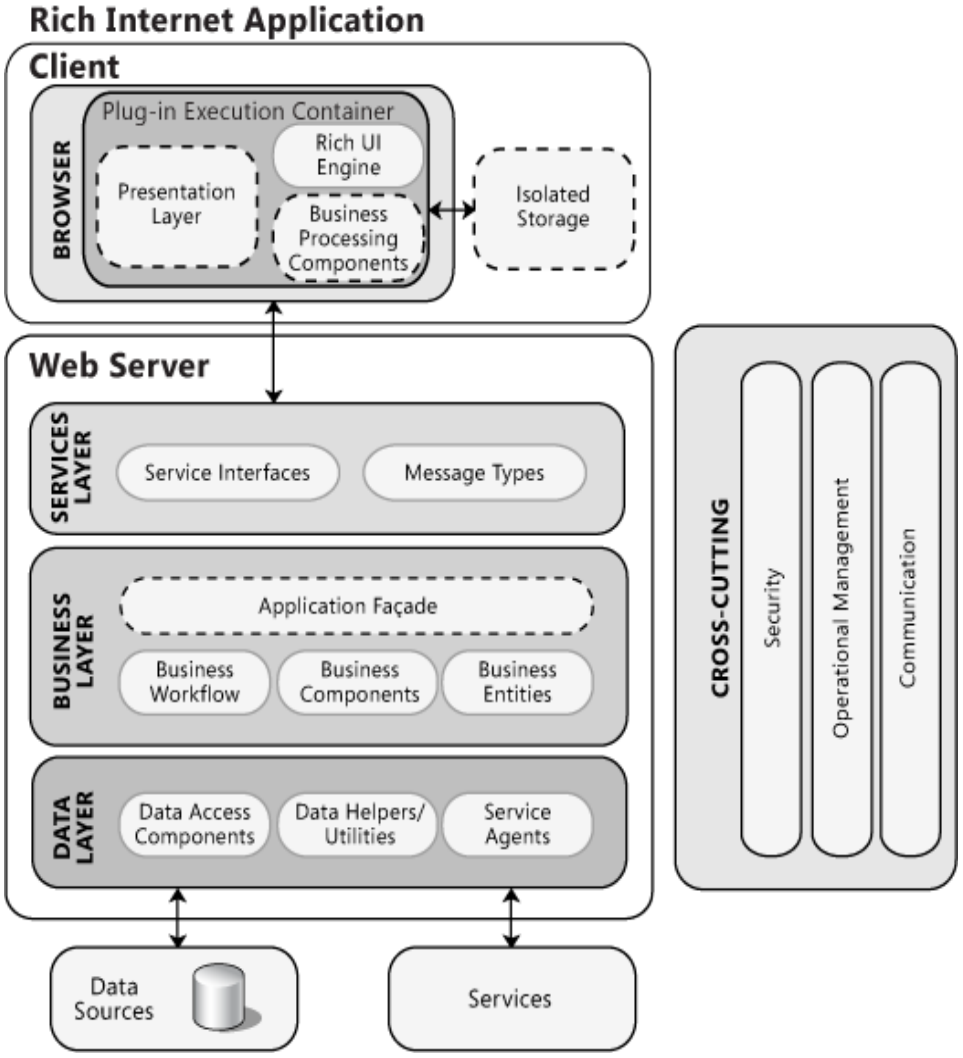



Ejemplo de lenguaje de patrones
Source: "SOA with REST" book

# Arquitecturas de referencia

Planos que proporcionan una estructura general para ciertos tipos de aplicaciones

- Pueden contener varios patrones
- Pueden ser un estándar *de-facto* en algunos dominios




Fuente: Microsoft Application Architecture Guide, 2nd Ed.

# Componentes desarrollados externamente

- Pilas tecnológicas o familias
  - MEAN (Mongo,Express,Angular,Node), LAMP (Linux,Apache,MySQL,PHP), ...
- Productos
  - COTS: Commercial Off The Self
  - FOSS: Free Open Source Software
    - ¡Cuidado con las licencias!
- Marcos de aplicación
  - Componentes de software reutilizables
- Plataformas
  - Proporcionan infraestructura completa para construir y ejecutar aplicaciones
    - Example: JEE, Google Cloud
- Librerías

# ADD - Attribute Driven Design

# ADD: Attribute-driven design

Define una arquitectura del software basada en ACs

Proceso de descomposición recursivo

En cada etapa, se eligen patrones y tácticas para satisfacer un conjunto de ACs

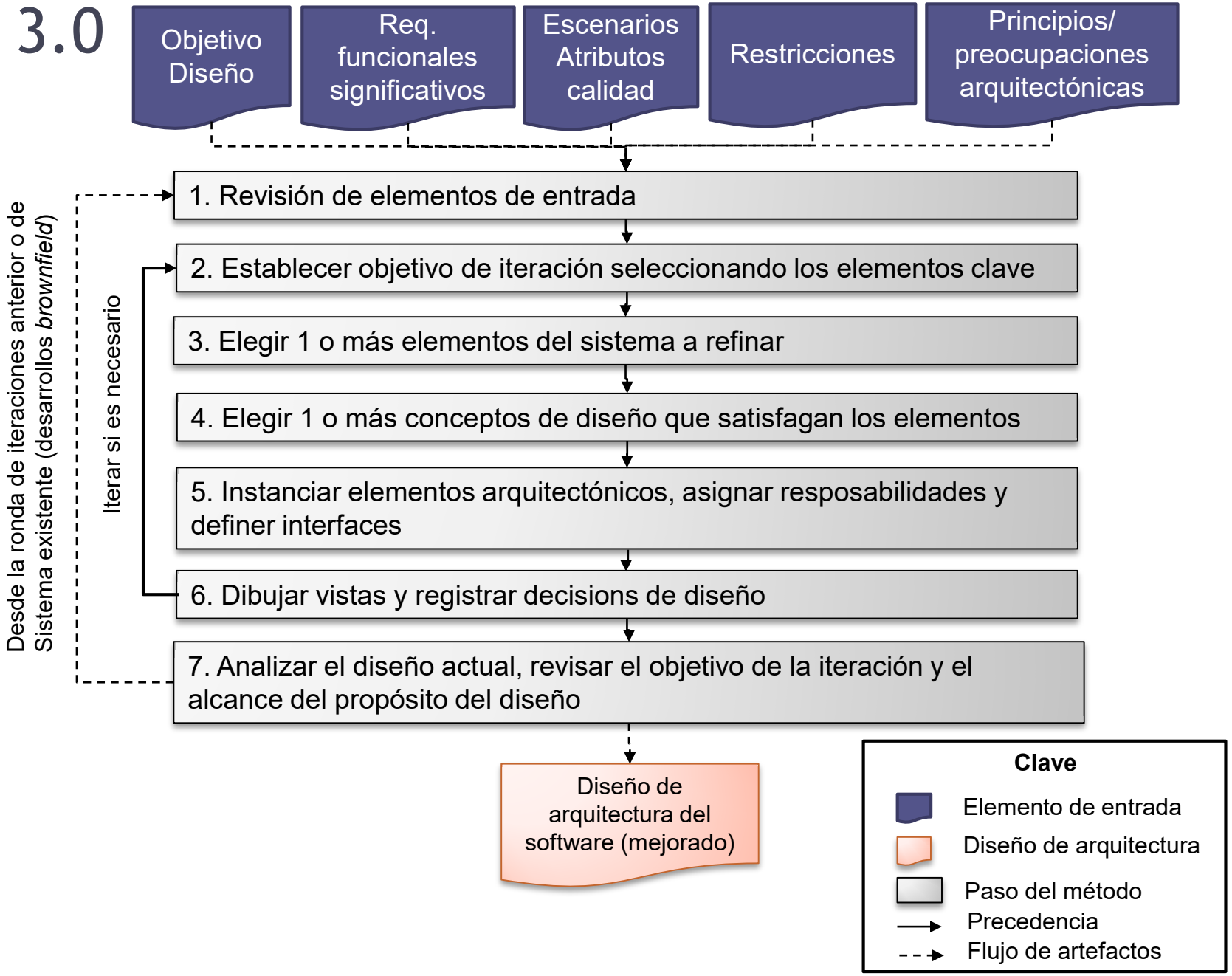
Entrada

- Requisitos AC
- Restricciones
- Req. funcionales significativos para la arquitectura

Salida

- Primeros niveles de descomposición modular
- Varias vistas del Sistema según se consideren apropiadas
- Conjunto de elementos con funcionalidad asignada e interacciones entre los elementos

# ADD 3.0


Objetivo Diseño
Req. funcionales significativos
Escenarios Atributos calidad
Restricciones
Principios/ preocupaciones arquitectónicas
1. Revisión de elementos de entrada
2. Establecer objetivo de iteración seleccionando los elementos clave
3. Elegir 1 o más elementos del sistema a refinar
4. Elegir 1 o más conceptos de diseño que satisfagan los elementos
5. Instanciar elementos arquitectónicos, asignar resposabilidades y definer interfaces
6. Dibujar vistas y registrar decisions de diseño
7. Analizar el diseño actual, revisar el objetivo de la iteración y el alcance del propósito del diseño
Desde la ronda de iteraciones anterior o de Sistema existente (desarrollos brownfield)
Iterar si es necesario
Diseño de arquitectura del software (mejorado)
Clave
Elemento de entrada
Diseño de arquitectura
Paso del método
Precedencia
Flujo de artefactos

# Registro decisiones arquitectónicas

Toda decisión de diseño es *suficientemente buena*
pero pocas veces óptima

Es necesario registrar la justificación y los riesgos

Cosas a registrar:

- ¿Cuál es la evidencia que justifica la decision?
- ¿Quién toma la decision?
- ¿Porqué se han tomado ciertos atajos?
- ¿Porqué se realizaron ciertos compromisos?
- ¿Qué suposiciones se han realizado?

| Clave | Decisión de diseño | Justificación y suposiciones |
|---|---|---|
| DA-1 | Introducir concurrencia (táctica) en TimeServerConnector y FaultDetectionService | La concurrencia debería ser introducida para ser capaz de recibir y procesar varios eventos simultáneamente |
| DA-2 | Utilizar un patron de mensajería mediante una cola de mensajes en la capa de comunicaciones | Aunque el uso de una cola de mensajes puede ir en contra del rendimiento impuesto por el escenario, será útil para dar soporte al escenario QA-3 |
| . . . | . . . | . . . |

# Registros decisiones arquitectónicas

Plantillas: https://adr.github.io/

Estructura básica:

- Título
  - Breve título descriptivo
- Estado
  - Propuesto, aceptado, reemplazado
- Contexto
  - Qué es lo que fuerza a tomar la decisión
  - Incluir alternativas
- Decisión
  - Decisión y justificación correspondiente
- Consecuencias
  - Impacto esperado de la decisión

Para borradores, puede ser útil utilizar RFCs (Request for comments)

# Incidencias arquitectónicas

# Incidencias arquitectónicas

- Riesgos
- Desconocidos
- Problemas
- Deuda técnica
- Diferencias de comprensión
- Erosión/Ir a la deriva

# Riesgos

- Riesgo = algo malo que podría ocurrir pero que todavía no ha ocurrido
- Riesgos deberían ser identificados y registrados
  - Riesgos pueden aparecer como parte de escenarios de AC
- Riesgos pueden ser mitigados o aceptados
  - Si es possible, identificar tareas de mitigación

# Tabla de valoración de riesgo

Valorar riesgos en 2 dimensiones:

- Impacto del riesgo
- Probabilidad de que ocurra

Pueden ser: bajo (1), medio (2), alto (3)

Probabilidad de que ocurra el riesgo

| Impacto esperado | Bajo (1) | Medio (2) | Alto (3) |
|---|---|---|---|
| Bajo (1) | 1 | 2 | 3 |
| Medio (2) | 2 | 4 | 6 |
| Alto (3) | 3 | 6 | 9 |

Ejemplo de valoración de riesgo

| Criterio de riesgo | Registro cliente | Realizar petición |
|---|---|---|
| Scalabilidad | 2 | 1 |
| Disponibilidad | 3 | 2 |
| Rendimiento | 4 | 3 |
| Securidad | 6 | 1 |
| Integridad datos | 9 | 1 |

# Desconocidos (unknowns)

Algunas veces no tenemos suficiente información sobre cómo una arquitectura puede satisfacer los requisitos

- Requisitos poco especificados
- Presuposiciones implícitas
- Requisitos cambiantes
- ...

Las evaluaciones arquitectónicas pueden ayudar a convertir los desconocidos en riesgos

# Problemas

- Problemas = cosas malas que ya han pasado
  - Tomar decisiones que no funcionan como se espera
  - Cambios en el contexto
    - Una decisión que era buena puede dejar de serlo
- Los problemas pueden arreglarse o ser aceptados
  - Los problemas que no se arreglan pueden generar una **deuda técnica** *(technical debt)*

# Deuda técnica

- Deuda adquirida cuando consciente o inconscientemente se toman decisiones de diseño equivocadas
  - Si se pagan los plazos la deuda es devuelta y no se crean más problemas
  - En caso contrario, se incurre en una penalización (interés)
    - Si no se puede pagar durante mucho tiempo, la deuda puede ser tan grande que se declara bancarrota
    - En términos software, significaría producto abandonado
- Varios tipos:
  - Deuda de código: Estilo de código malo o inconsistente
  - Deuda diseño: *Malos olores* de diseño
  - Deuda de prueba: Falta de pruebas, poca cobertura,...
  - Deuda documentación: Aspectos importantes sin documentación, documentación no actualizada,...

# Diferencias de comprensión

- Aparecen cuando lo que piensan los stakeholders sobre la arquitectura no encaja con el diseño
  - Las arquitecturas evolucionan rápidamente y las diferencias aparecen rápidamente y sin previo aviso
- Diferencias pueden afrontarse con formación
  - Presentando la arquitectura a los *stakeholders*
  - Realizando preguntas a los *stakeholders*

# Deterioro arquitectónico

- Diferencia entre la arquitectura diseñada y la arquitectura del sistema construido
  - El sistema implementado casi nunca se parece al sistema que el arquitecto se imagina
  - Sin vigilancia, la arquitectura puede ir derivando y alejándose del diseño planificado hasta que un día apenas se parezcan
- Código arquitectónicamente evidente puede mitigar esta diferencia

# Evolución del contexto

Ocurre cuando aspectos claves del contexto cambian después de haber tomado una decision de diseño

- Es necesario revisar continuamente los requisitos
- Arquitecturas evolutivas

# Evaluación de arquitecturas

# Evaluación de la arquitectura

ATAM (Architecture Trade-off Analysis Method)

Método para evaluar arquitecturas del software

Versión simplificada:

- Presenta aspectos clave de negocio
- Presentar arquitectura
- Identificar enfoques de la arquitectura
- Generar árbol de utilidad de Atributos de Calidad
- Analizar enfoques arquitectónicos
- Presenta resultados

# Cost Benefit Analysis Method (CBAM)

1. Elegir escenarios y estrategias arquitectónicas
2. Valorar beneficios para atributos de calidad
3. Cuantificar beneficios de estrategias arquitectónicas
4. Cuantificar costes e implicaciones de las estrategias
5. Calcular la deseabilidad de cada opción
6. Tomar decisiones de diseño arquitectónico

Fin