



Universidad de Oviedo



Rol del arquitecto y Atributos de calidad



Curso 2019/2020

Jose Emilio Labra Gayo

Contenidos

Rol del arquitecto

Atributos de calidad

Escenarios de atributos de calidad

Conceptos de diseño:

Arquitecturas de referencia, patrones, estilos y tácticas

ADD: attribute-driven design

Incidencias de arquitectura

Evaluación de arquitecturas (ATAM)

Role del arquitecto de software



Rol del arquitecto

Expectativas sobre un arquitecto

- Tomar decisiones arquitectónicas

- Analizar continuamente la arquitectura

- Estar al día de las tendencias actuales

- Asegurar cumplimiento decisiones existentes

- Experiencia diversa

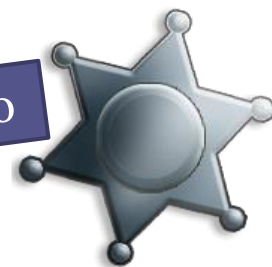
- Conocimiento del dominio de negocio

- Poseer habilidades interpersonales

- Comprender y navegar en política empresarial

Leyes de arquitectura del software

Arquitecto de software es un rol, no un rango



Tomar decisiones arquitectónicas

Definir decisiones de arquitectura y principios de diseño

El arquitecto debe guiar las decisiones tecnológicas

Mantener registros de decisiones

Analizar puntos a favor y en contra



Analizar continuamente la arquitectura

Revisar continuamente tecnología y arquitectura

Ser responsable del éxito técnico del proyecto

Estar al tanto de posible deterioro estructural

Perseguir la consistencia

Organizar código en paquetes, directorios, módulos,...

Definir límites, principios, guías,...

Incluir entornos de prueba y entrega en proyectos



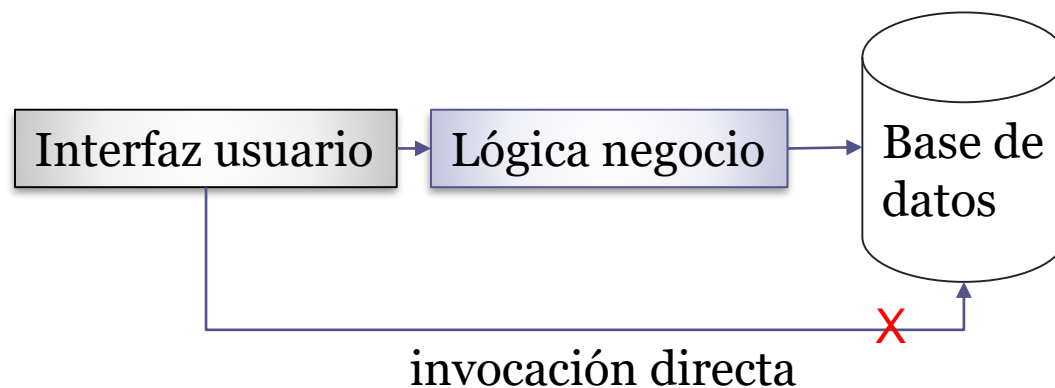
Asegurar cumplimiento de decisiones

Los arquitectos normalmente imponen restricciones

Ejemplo:

Acceso a base de datos desde interfaz de usuario

Los desarrolladores se las podrían saltar

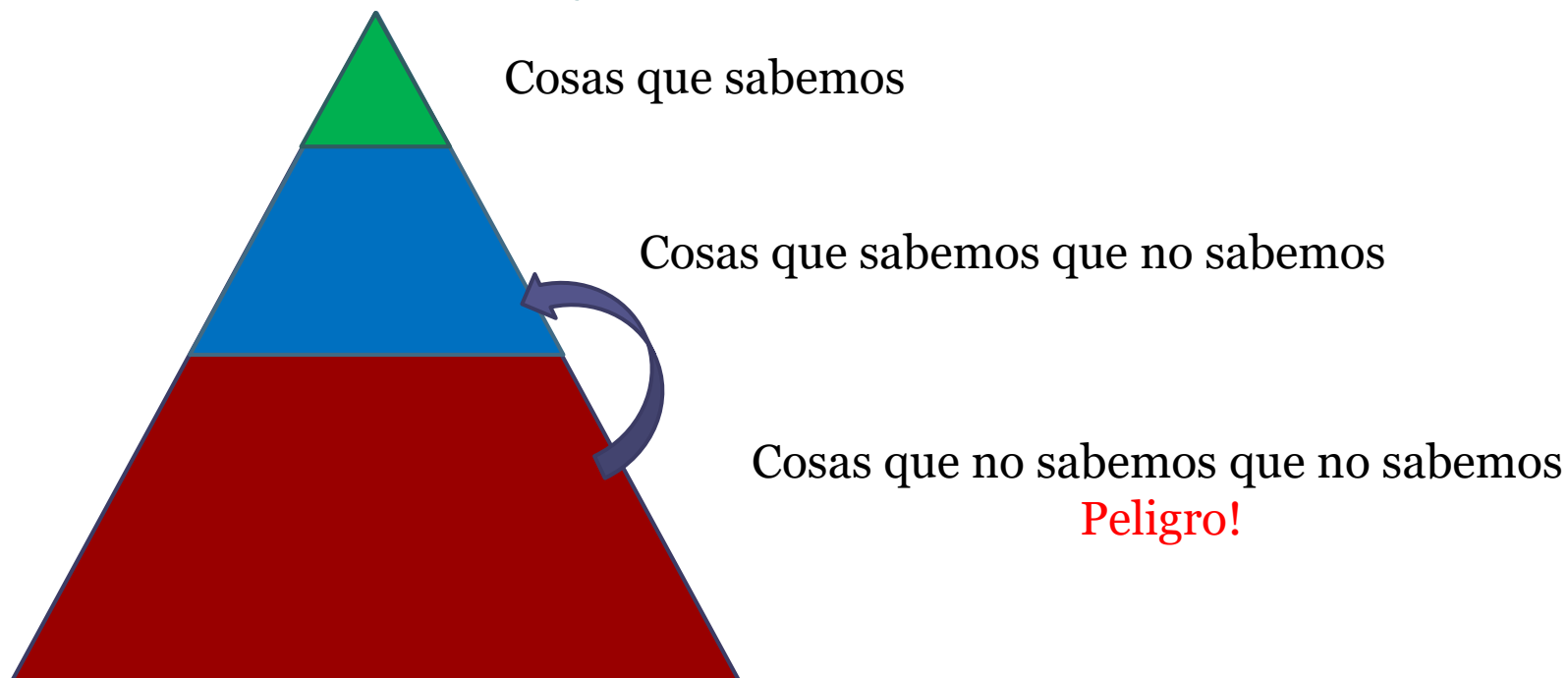


Estar al día de las tendencias

Conocer últimas tendencias tecnológicas e industriales

Las decisiones de un arquitecto pueden tener larga duración y ser muy costosas

Conocer lo que sabes y lo que sabes que no sabes



Experiencia diversa

Estar expuesto a múltiples y diversas tecnologías, marcos, plataformas, entornos, etc.

No quiere decir ser un experto en todas ellas
...pero al menos estar familiarizado con varias tecnologías

Amplitud técnica mejor que profundidad técnica



Conocimiento dominio de negocio

Se espera que el arquitecto tenga un cierto conocimiento del dominio de negocio

Comprensión del problema de negocio, los objetivos y los requisitos

Comunicarse de forma efectiva con ejecutivos y usuarios del dominio utilizando su lenguaje



Habilidades interpersonales

Arquitecto del software = líder

Habilidades de trabajo en equipo y liderazgo

Liderazgo técnico

Ser inclusivo y colaborador

Ayudar a desarrolladores a comprender la estructura general (*the big picture*)

Participar en desarrollo

Formar parte de la entrega

Comprensión de bajo nivel

Codificación como parte del rol

Revisiones de código y tutorización



"no importa lo que te digan, siempre es un problema de personas", G. Weinberg

Comprender y navegar la política

Comprender el clima político de la empresa y ser capaz de navegar la política empresarial

Decisiones arquitectónicas afectan a stakeholders

Dueños de producto, gestores de proyecto, personas de negocio, desarrolladores, etc.

Casi cualquier decisión tomada por un arquitecto va a ser discutida y puesta en duda

Habilidades de negociación son necesarias

Presentar y defender la arquitectura



Leyes de arquitectura del software

Todo en arquitectura del software es un equilibrio entre varias soluciones

—1ª Ley arquitectura Software



Si un arquitecto cree que ha descubierto algo que no es un equilibrio entre varias soluciones, lo más probable es que no lo haya identificado todavía
—Corolario 1

Atributos de calidad

Características de la arquitectura

Tipos de requisitos

Los requisitos pueden clasificarse en:

Requisitos funcionales: indican lo que debe hacer el sistema, cómo debe comportarse o reaccionar a un estímulo en tiempo de ejecución

Requisitos de atributos de calidad. Anotan (cualifican) requisitos funcionales

Ejemplos: Disponibilidad, modificabilidad, usabilidad, seguridad,...

También llamados: requisitos no-funcionales

Restricciones. Decisiones de diseño que ya han sido tomadas

Requisitos funcionales

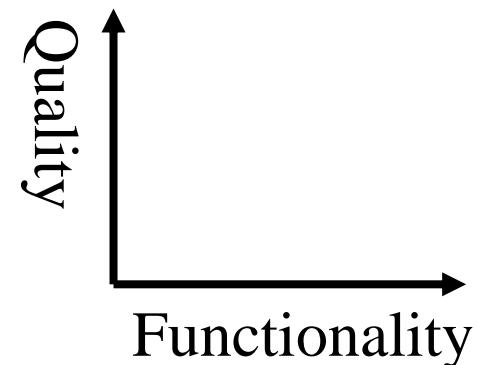
Funcionalidad = capacidad del Sistema para realizar el trabajo para que se pretende que haga

La funcionalidad tiene una relación extraña con arquitectura:

No determina la arquitectura

Si el único requisito fuese la funcionalidad, el Sistema podría existir como un único modulo monolítico sin ninguna estructura interna

La funcionalidad y los atributos de calidad son ortogonales



Atributos de calidad (AC)

Los atributos de calidad cualifican la funcionalidad

Si un requisito funcional fuese "*cuando el usuario pulse el botón verde, un diálogo con opciones aparece*" entonces:

- Un AC de rendimiento describiría la velocidad a la que debe aparecer
- Un AC de disponibilidad describiría cuándo podría fallar esta opción o con qué frecuencia sería reparada
- Un AC de usabilidad describiría cómo de fácil es aprender esta función

Los atributos de calidad **sí** influyen en la arquitectura

¿Qué es la calidad?

Grado en que un Sistema satisface las necesidades declaradas o implícitas de las personas interesadas proporcionando valor

- Grado (no Booleano)
- Calidad = Encaje en un propósito (necesidades de *stakeholders*)
- Satisfacer requisitos (declarados o implícitos)
- Proporcionar valor

Algunas definiciones de calidad

https://en.wikipedia.org/wiki/Software_quality

ACs y soluciones de compromiso

Todos los ACs son buenos

...pero el valor depende del Proyecto y *stakeholder*

"Mejor calidad"...para qué?, para quién?

ACs no son independientes

Algunos ACs pueden entrar en conflicto

Qué es lo más importante?

Ejemplo: Sistema muy seguro puede ser menos usable

Siempre hay un precio



¡No hay sistema o arquitectura perfecto!

Especificando Atributos Calidad

2 consideraciones:

Los ACs por sí mismos no son suficientes

No son operacionales

Ejemplo: No aporta mucho decir que un Sistema debe ser modificable, o tener alta disponibilidad

El vocabulario para describir ACs es muy variado

Es necesario describir cada atributo por separado

Escenarios Calidad: Forma común de especificar requisitos de AC

Escenario de Calidad

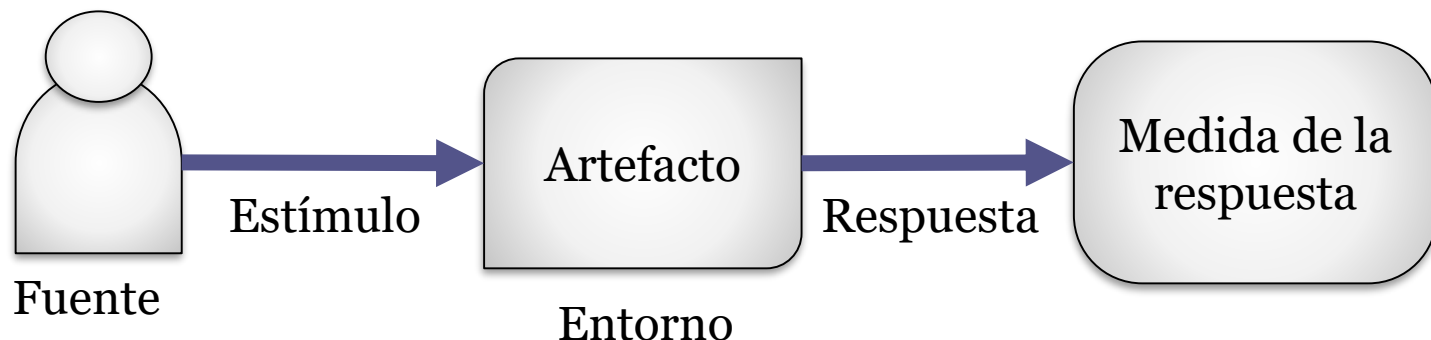
Describe un estímulo de un Sistema y una respuesta medible a dicho estímulo

Estímulo = evento iniciado por una persona o sistema

El estímulo genera una respuesta

Debe ser medible

La respuesta debe ser visible externamente y medible



Componentes de escenario calidad

Fuente: Persona o Sistema que inicia el estímulo

Estímulo: Evento que requiere que responda el sistema

Artefacto: Parte del sistema o el sistema completo

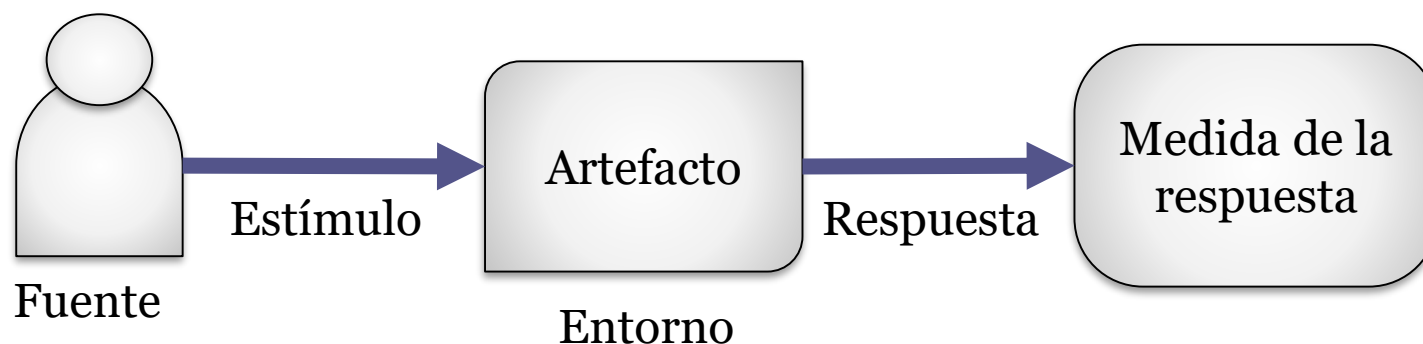
Respuesta: Acción visible externamente

Medida de respuesta: Criterio de éxito para el escenario

Debe ser específico y medible

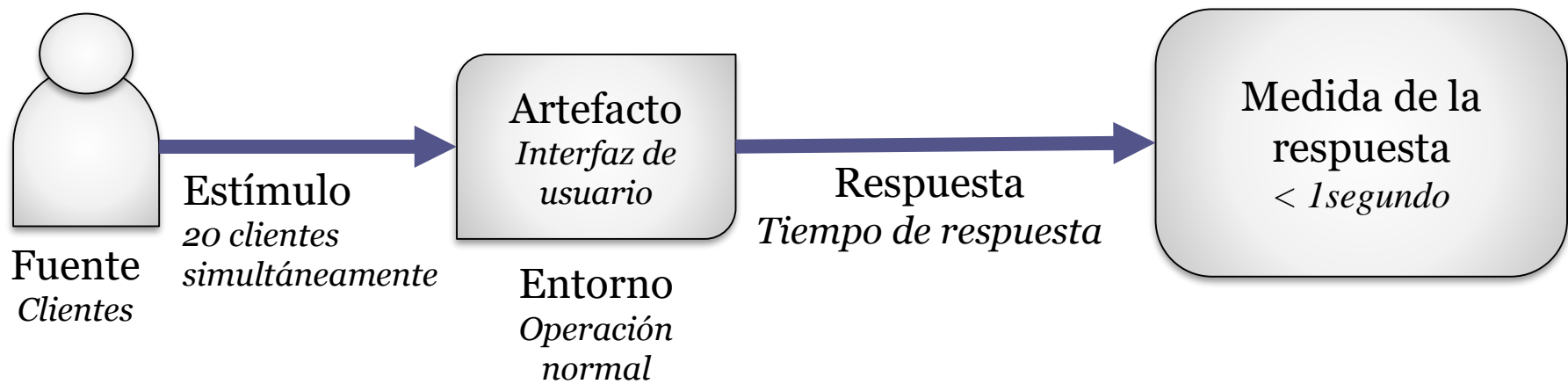
Entorno: Circunstancias operativas

Siempre debe ser definido (incluso si es "normal")

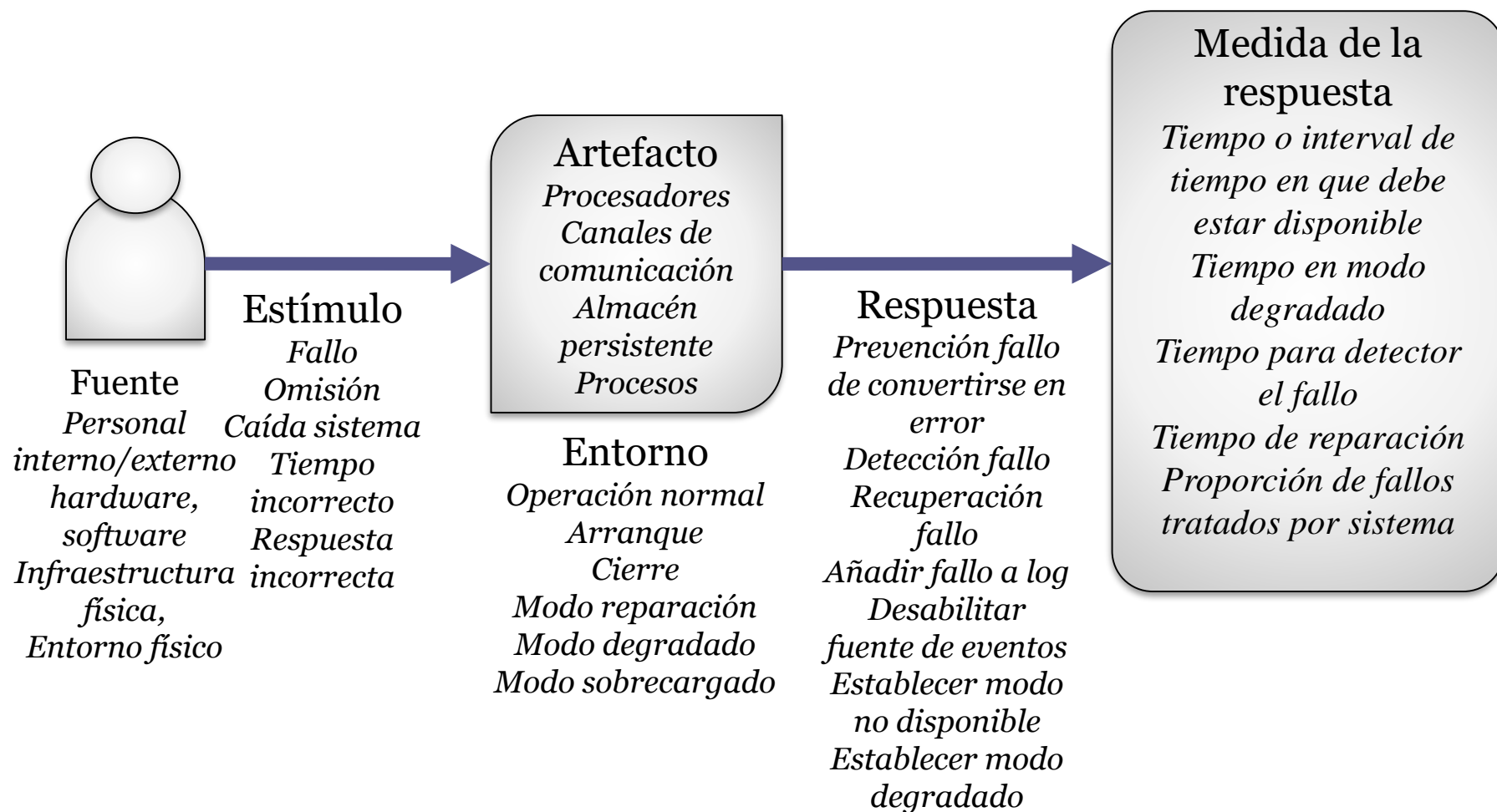


Escenario de calidad, ejemplo 1

Rendimiento: *Si hay 20 clientes simultáneamente, el tiempo de respuesta debería ser menos que 1 segundo en circunstancias normales*



Escenario calidad para disponibilidad



Tipos de escenarios de calidad

Escenarios de uso

El Sistema es utilizado (cualquier caso de uso o función del Sistema que se ejecuta)

Describe cómo se comporta el Sistema en dichos casos

Tiempo de ejecución, velocidad de respuesta, consume de memoria, *throughput*,...

Escenarios de cambio (o modificación)

Cambio en cualquier componente dentro del Sistema, en entorno o la infraestructura operacional

Escenarios de fallo:

Alguna parte del Sistema, infraestructura o entorno falla

Priorizando escenarios de calidad

Los escenarios deben ser priorizados

2 valores (Low/Medium/High)

Cómo es de importante para el éxito (cliente)

Cómo de difícil es de alcanzar (arquitecto)

Ref	AQ	Scenario	Priority
1	Disponibilidad	Cuando la base de datos no responda, el Sistema debería registrar el fallo en el log y responder con datos anteriores durante 3 seg.	High, High
2	Disponibilidad	Un usuario que busca elementos de tipo X recibe una lista de Xs el 99% del tiempo como media a la largo del año	High, Medium
3	Escalabilidad	Nuevos servidores pueden ser añadidos durante la ventana de mantenimiento (en menos de 7 horas)	Low, Low
4	Rendimiento	Un usuario puede ver los resultados de búsqueda en menos de 5 segundos cuando el Sistema tiene una carga de 2 búsquedas por seg	High, High
5	Fiabilidad	Las actualizaciones a elementos externos de tipo X deberían reflejarse en la aplicación dentro de 24 horas del cambio	Low, Medium

Identificando Atributos de calidad

¿Cómo encontrar los Atributos de calidad?

La mayoría de las veces los no están explícitos

Pueden ser mencionados de pasada en req. funcionales

Normalmente implícitos o mencionados sin mucha consideración

Arquitecto del software debe tratar de identificarlos

Talleres de atributos de calidad

Reuniones con stakeholders para identificar y priorizar los atributos y escenarios de calidad

Listas formales

ISO25010

Wikipedia:

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Atributos de calidad habituales

Disponibilidad

Modificabilidad

Rendimiento

Seguridad

Testabilidad

Mantenibilidad

Usabilidad

Escalabilidad

Interoperabilidad

Portabilidad

Cambiabilidad

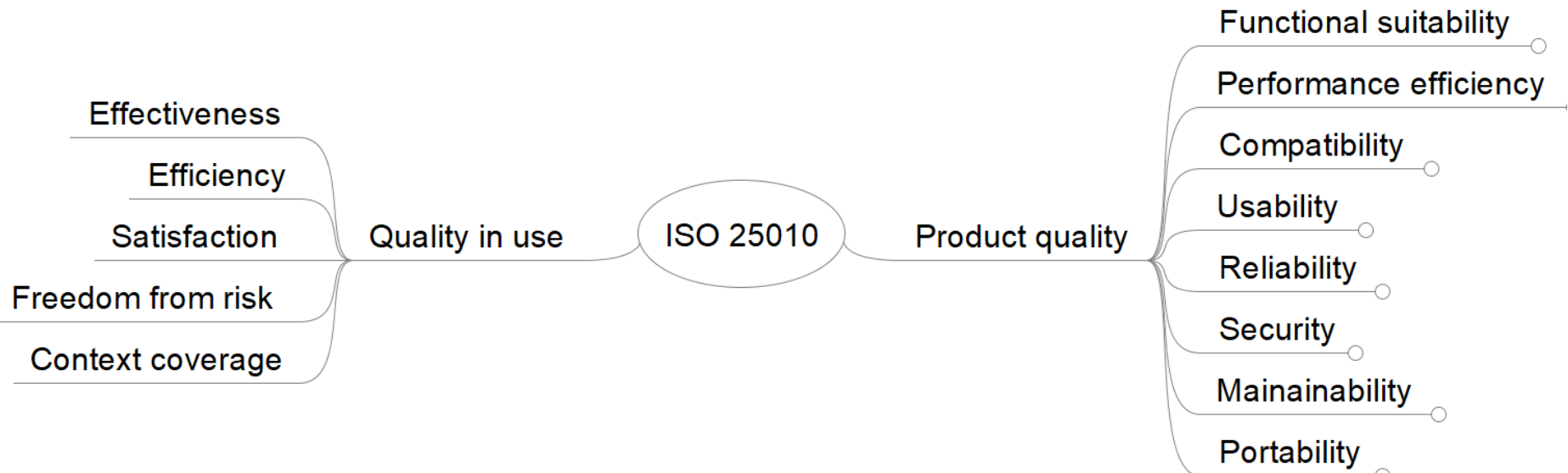
. . .

ISO-25010 Software Quality Model

Systems and software Quality Requirements and Evaluation (SQuaRE)

2 partes:

- *Quality in-use*
- *Product quality*

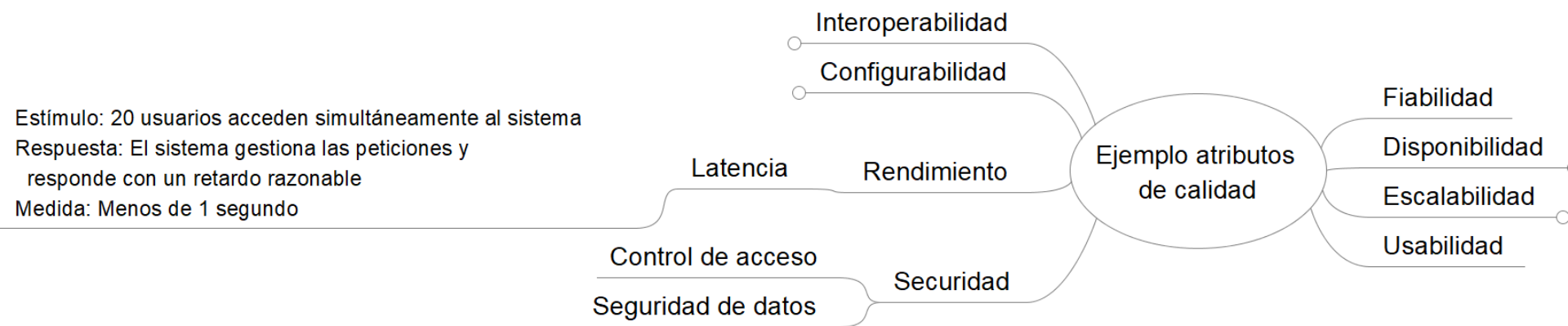


<https://arquisoft.github.io/Iso25010QualityAttributes.html>

Árbol de Atributos de calidad

Representaciones Mindmap pueden ser útiles para visualizar Atributos de calidad

Permiten seleccionar y ampliar atributos bajo demanda



Conceptos de diseño

Tácticas, estilos, patrones, arquitecturas de referencia

Tácticas

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

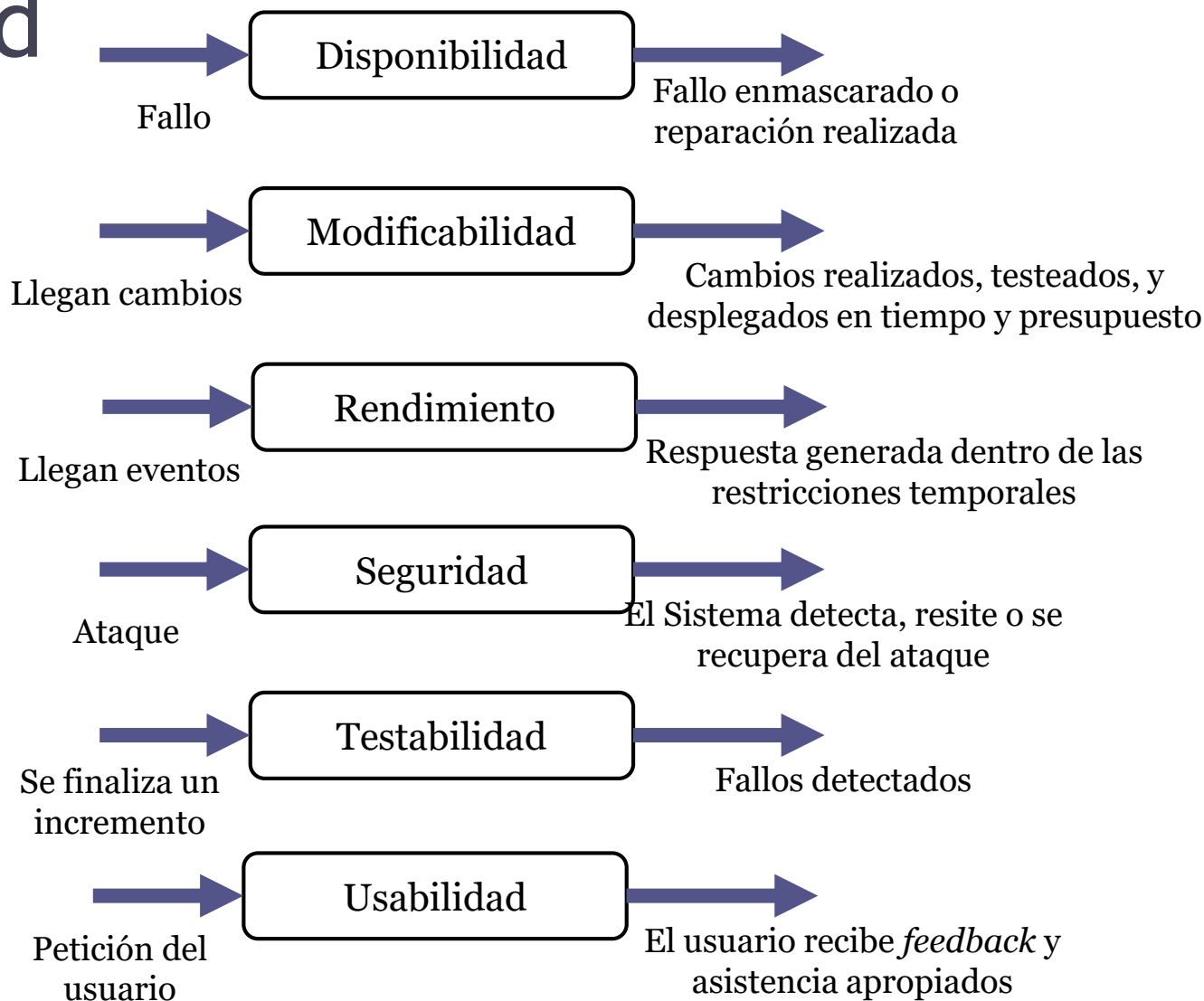
Las tácticas se enfocan en la respuesta a un Atributo de calidad

Pueden chocar con otros atributos de calidad

Las tácticas intentan controlar respuestas a estímulos



Tácticas dependen del atributo de calidad



¿Dónde podemos encontrar tácticas?

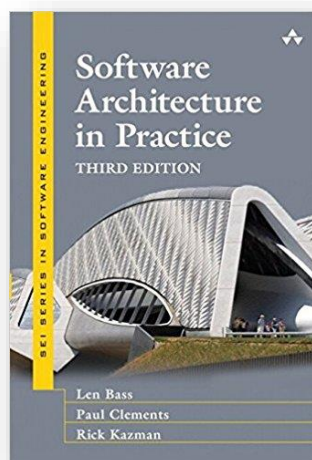
Propia experiencia del arquitecto

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:

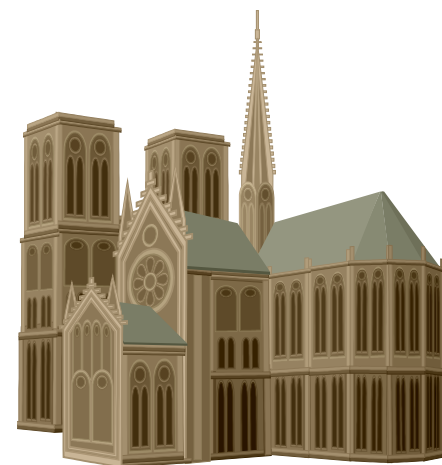
Elementos: Componentes que contienen funcionalidad

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

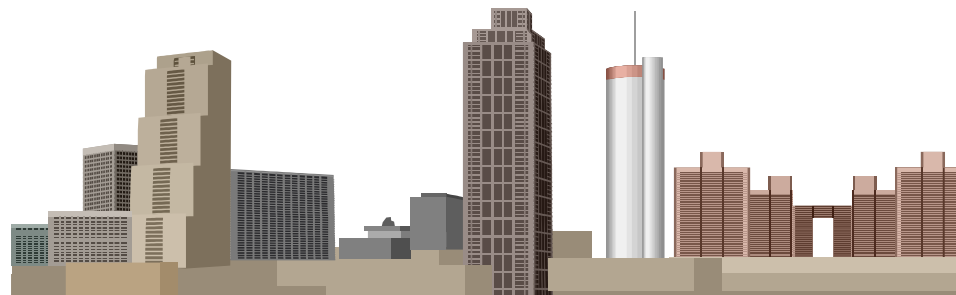
En práctica, los estilos puros se dan pocas veces
Normalmente, los sistemas se desvían de estilos puros...

...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

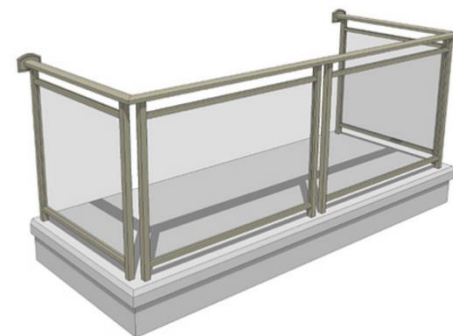
Ejemplo: Layers

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 patrón 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

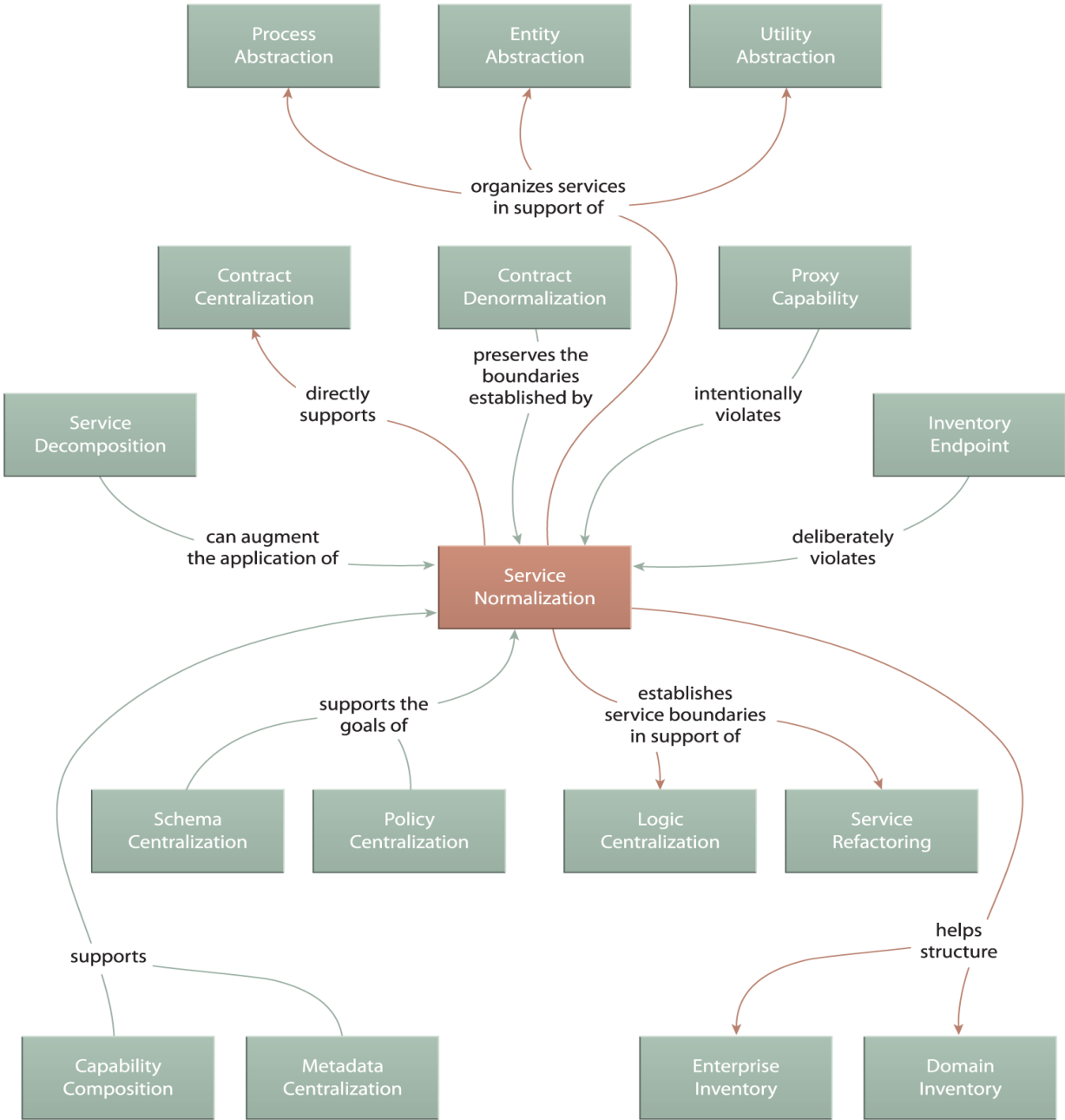
Lenguaje de patrones

- Un catálogo de patrones completo sobre un tema

- Objetivo: documentar todas las posibilidades

- Normalmente incluyen relaciones entre patrones

- Mapa gráfico



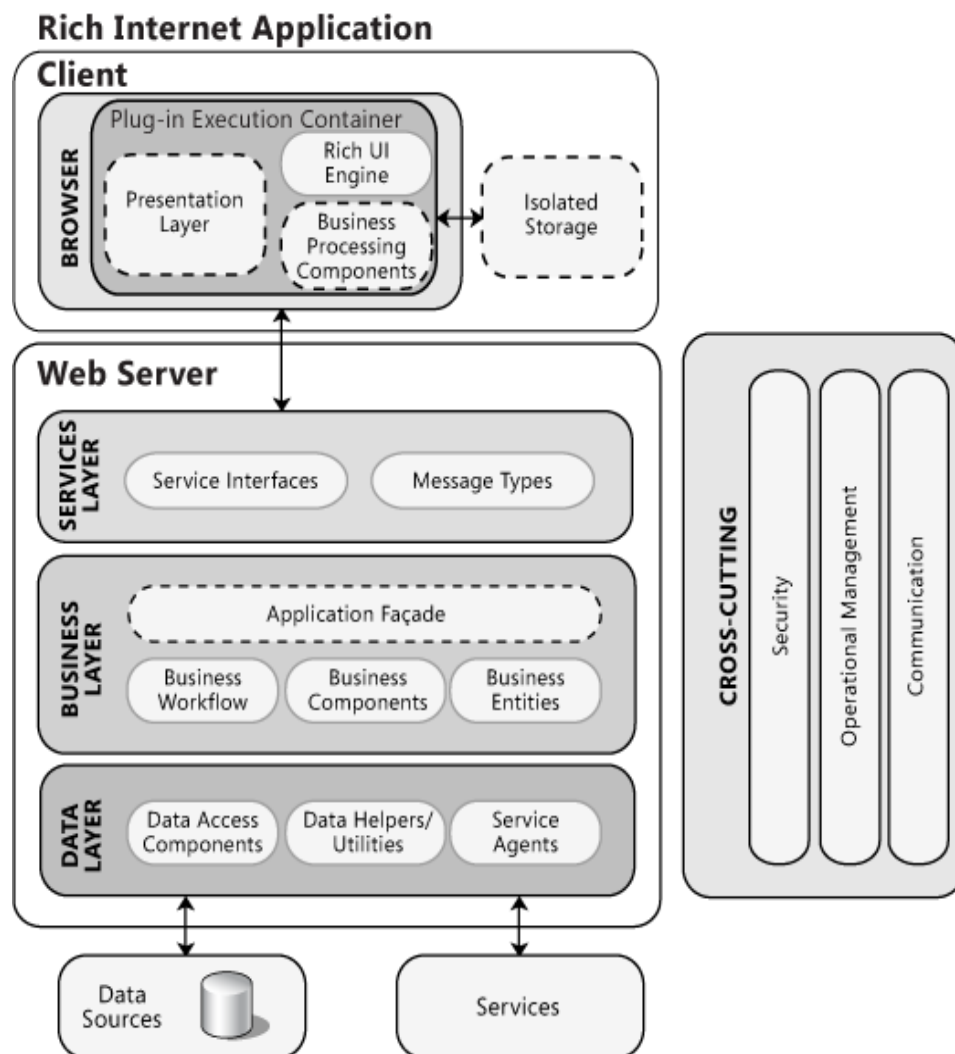
Ejemplo de lenguaje de patrones
Source: "SOA with REST" book

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



Componentes desarrollados externamente

Pilas tecnológicas o familias

MEAN (Mongo, Express, Angular, Node), **LAMP** (Linux, Apache, MySQL, PHP), ...

Productos:

COTS: Commercial Off The Shelf

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

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

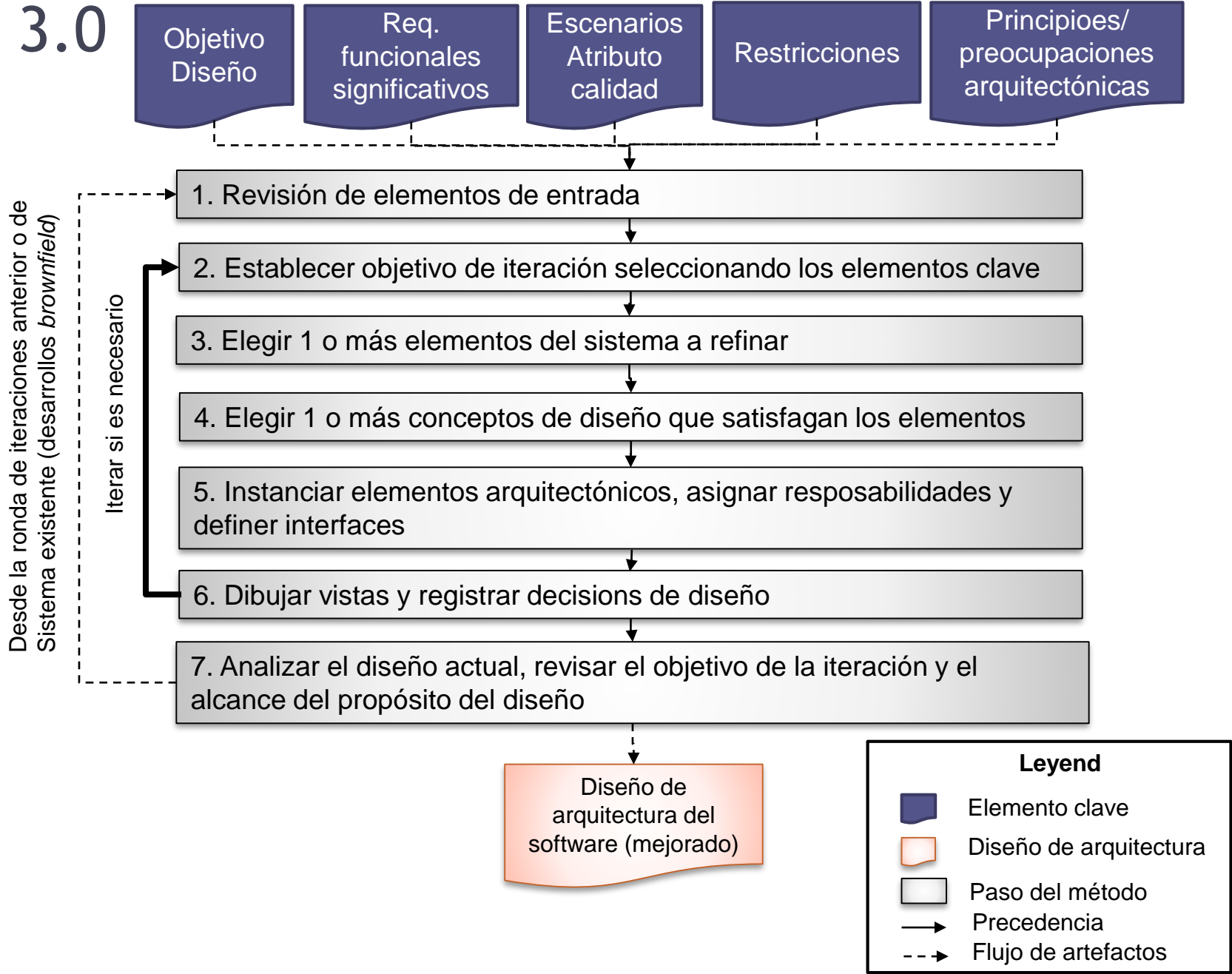
Entrada

- 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



Registro de 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 decisión?

¿Quién toma la decisión?

¿Porqué se han tomado ciertos atajos?

¿Porqué se realizaron ciertos compromisos?

¿Qué suposiciones se han realizado?

Clave	Decisión de diseño	Justificación y suposiciones
AC-1	Introducir concurrencia (táctica) en TimeServerConnector y FaultDetectionService	La concurrencia debería ser introducida para ser capaz de recibir y procesar varios eventos simultáneamente
AC-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
...

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

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 son cosas malas que ya han pasado

Aparecen cuando se toman decisiones de diseño que simplemente no funcionan de la forma deseada

También pueden aparecer por cambios en el contexto

Una decisión que era una idea buena puede dejar de tener sentido

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 un 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 decisión 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 un árbol de utilidad de ACs
- 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