



(1306)

ARQUITECTURA DE SOFTWARE

I.I Introducción al Curso

Agenda

- ¿Quien es el profesor?
- ¿De que se trata la materia?
- Contenido Temático
- Competencias Transversales y Resultados de Aprendizaje Esperados
- Actividades (Estrategias Pedagógicas)
- Parcelación (Evaluación)
- Reglas
- Medios de comunicación
- Bibliografía
- Campus Virtual
- Motivación
- Importancia
- ¿Quiénes son los estudiantes?

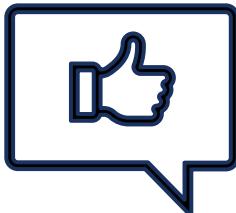


¿QUIEN ES EL PROFESOR?

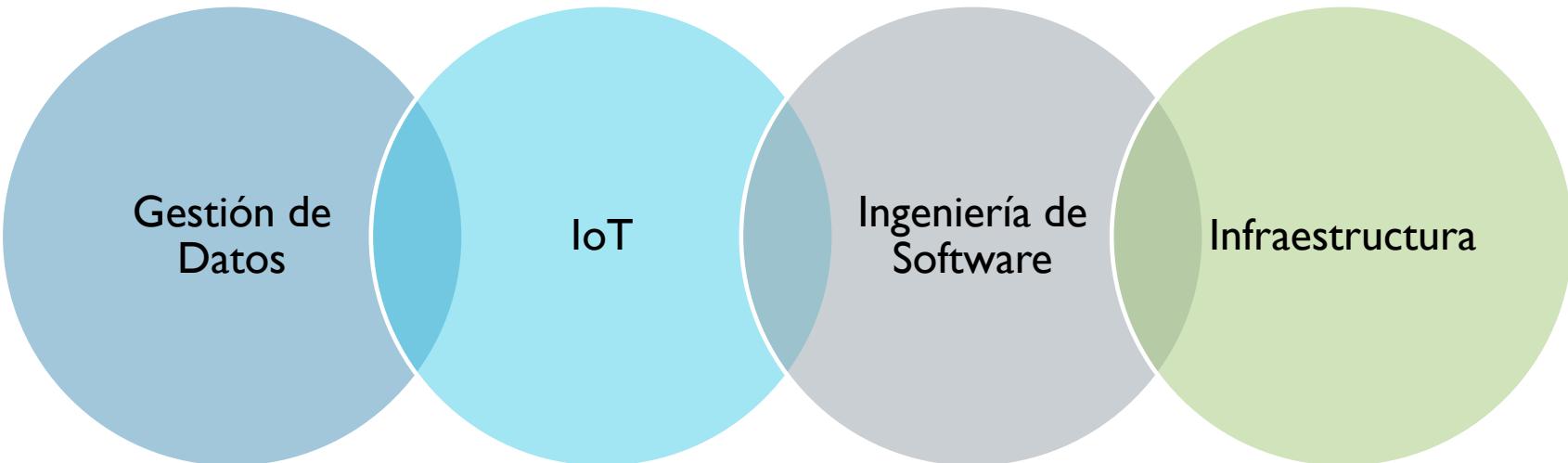
■ Andrés Armando Sánchez Martín:

Ingeniero de Sistemas de la Universidad Católica de Colombia. Maestría en Ingeniería de Sistemas y Computación de la Pontificia Universidad Javeriana, más de 12 años de experiencia profesional en proyectos de tecnología, tanto en infraestructura, gestión de datos, desarrollo de software y servicios web.

Profesor de Catedra PUJ, y con 10 años de experiencia docente en diferentes IES

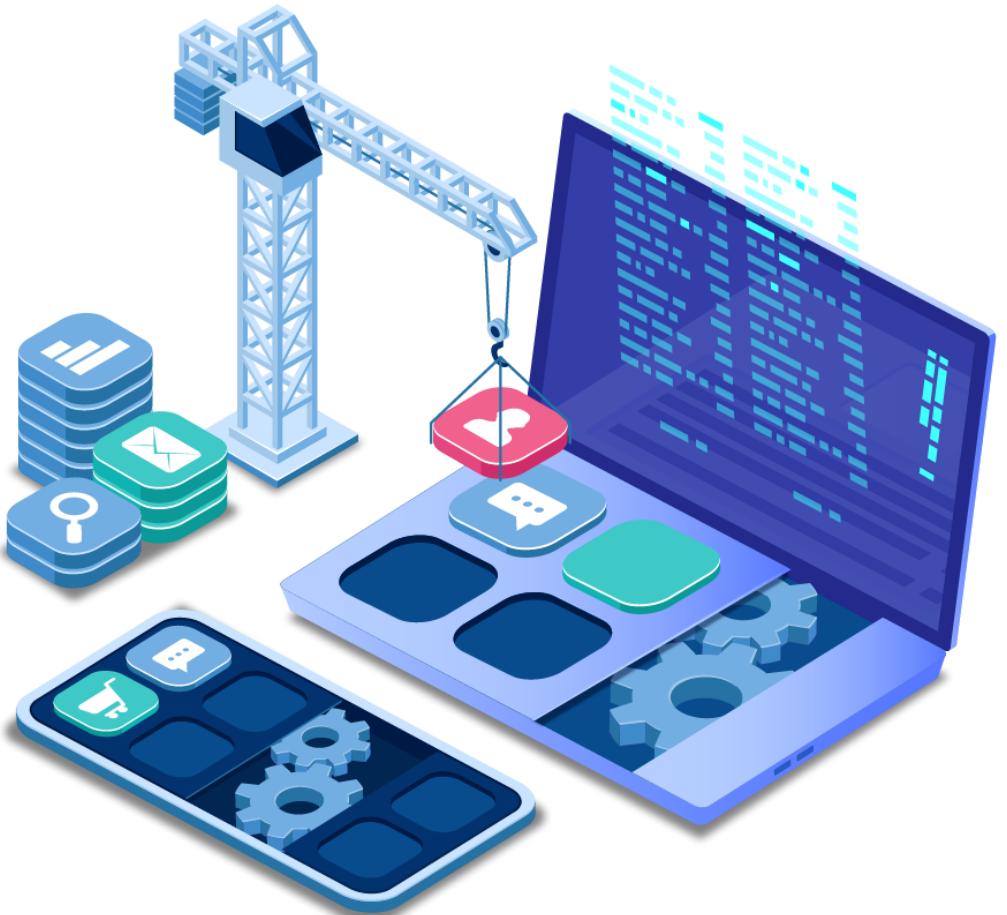


Intereses en:



¿DE QUE SE TRATA LA MATERIA?

Este curso propicia un ambiente de aprendizaje en el cual se aplican metodologías, herramientas y estrategias enfocadas en apoyar el diseño e implementación de arquitecturas de sistemas grandes cumpliendo con los atributos de calidad. Lo cual tiene como objetivo que el estudiante adquiera las competencias necesarias para analizar, diseñar e implementar arquitecturas de software que permitan incrementar la productividad y competitividad de las organizaciones. La asignatura cubre temáticas como atributos de calidad, estilos y patrones arquitectónicos, procesos de diseño, arquitecturas de referencia y documentación, entre otros. La asignatura tiene tres componentes metodológicos para el desarrollo de los temas: aprendizaje directivo, autoaprendizaje y un componente práctico. Adicionalmente, los estudiantes tendrán la oportunidad de realizar talleres en clase con el fin de resolver dudas y afianzar los conceptos.



Objetivos de Formación:

- Presentar los conceptos fundamentales de arquitectura de software para contextualizar las problemáticas que aborda, y su relación con otras temáticas de la ingeniería.
- Presentar los modelos de calidad de software que se aplican a la arquitectura de software para que los estudiantes seleccionen un modelo de referencia.
- Presentar los estilos y patrones arquitectónicos relacionándolos con los atributos de calidad que ofrecen, para identificar diseños de referencia y sus atributos de calidad.
- Presentar los métodos de diseño arquitectural enfatizando en uno de ellos, para que sea usado en los procesos de construcción de software.
- Presentar marcos de trabajo, herramientas y técnicas de implementación de arquitecturas de referencia para desarrollar un proyecto de aplicación



CONTENIDO TEMATICO

Andrés Armando Sánchez Martín

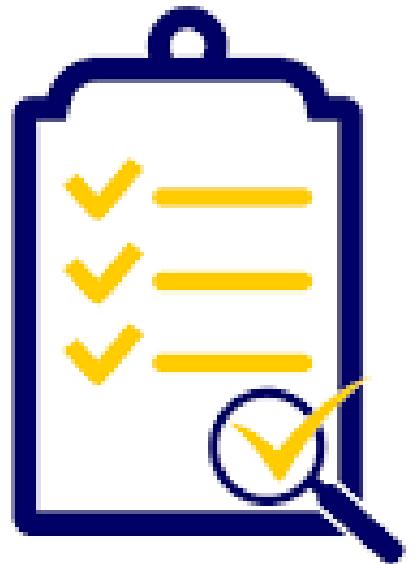
Conceptos fundamentales de AS

Atributos de calidad y estilos arquitecturales

Proceso de diseño de una arquitectura de software

Validación de arquitecturas de software

Implementación de AS



COMPETENCIAS TRANSVERSALES Y RESULTADOS DE APRENDIZAJE ESPERADOS

Las competencias transversales que aplican:

2.1 RAZONAMIENTO ANÁLITICO Y SOLUCIÓN DE PROBLEMAS:

- Analizar las diferentes alternativas de solución para un problema, bajo criterios de riesgo, costo-beneficio, restricciones, impactos, etc. (2)
- Escoger la alternativa más adecuada para modelar el problema. (2)

4.3 CONCEPCIÓN, APLICACIÓN DE LA INGENIERÍA Y GESTIÓN DE SISTEMAS:

- Aplicar los procedimientos necesarios para transitar de un conjunto de requisitos a una solución funcional (3)

4.4 DISEÑO:

- Aplicar los procedimientos necesarios para hacer diseño disciplinar y multidisciplinario (3)
- Aplicar los procedimientos necesarios para evaluar la calidad de un diseño disciplinar y multidisciplinario (3)
- Aplicar la utilización del conocimiento y los modos de pensamiento para el diseño de soluciones informáticas (3)

4.5 IMPLEMENTACIÓN

- Aplicar los elementos y consideraciones que hacen sostenible un proceso de implementación de una solución de ingeniería (3)

Al final del curso el estudiante debe estar en capacidad de:

1. **Comparar** estructuras, estilos, patrones y vistas de una arquitectura de software, así como atributos de calidad. (Disciplinar 1 y 2) (CDIO 2.1)
2. **Elegir** entre las alternativas arquitecturales para modelar adecuadamente un sistema dado (Disciplinar 2 y 3) (CDIO 2.1).
3. **Aplicar los procedimientos necesarios para** alinear la arquitectura de un sistema en su nivel de tecnología, información y aplicación. (Disciplinar 3 y 4) (CDIO 4.3 y 4.4)
4. Aplicar los procedimientos necesarios para la integración del sistema desarrollado con otros sistemas, protocolos, aplicaciones y plataformas. (Disciplinar 3 - 5) (CDIO 4.4)
5. **Aplicar** en el sistema a desarrollar los últimos estándares y metodologías disponibles (UML, XML, IEEE-1471). (Disciplinar 3 y 5) (CDIO 4.3 y 4.4)
6. Aplicar los elementos y consideraciones en la implementación de arquitecturas de software (Disciplinar 5) (CDIO 4.5)

ACTIVIDADES (ESTRATEGIAS PEDAGÓGICAS)

Durante el curso se utilizarán 3 estrategias. La primera será el aprendizaje directivo mediado por clases magistrales interactivas. La segunda será el aprendizaje basado en problemas mediante talleres individuales o grupales, donde se expone a los estudiantes a problemas de diferente nivel de complejidad, que refuerzan los conceptos y son útiles para la elaboración de los proyectos del curso. La tercera será el aprendizaje por proyectos, donde los contenidos del curso se irán trabajando grupalmente en equipos de estudiantes que resolverán en entregas un proyecto cuyo enunciado es dado por el profesor. Para la construcción de modelos y la generación de código, los estudiantes cuentan con los laboratorios de la carrera que podrán usar fuera de las horas presenciales.

Quices

Talleres

Presentaciones

Laboratorios

Parciales

Proyecto



PARCELACIÓN (EVALUACIÓN)

Las estrategias de evaluación están centradas en la valoración de los resultados de aprendizaje esperado de la asignatura; las cuales pueden ser formativas, que suscitan la comprensión y construcción de conocimiento, y sumativas, las cuales incluyen porcentajes de evaluación con el fin de corroborar el logro de los aprendizajes y el desarrollo de las competencias en los estudiantes. Las estrategias de evaluación de la asignatura son:

Evaluación de Conocimientos Teóricos en
Arquitecturas de Software:

Parcial No.1 20%

Parcial No.2 20%

Evaluación de Conocimientos Prácticos en
Arquitecturas de Software:

Proyecto Semestre:

Evaluación de
Preparación, seguimiento
de la clase y
responsabilidad:

Tareas & Quices &
Laboratorios 20%

Entrega No.1: SAD
arquitectura lógica sin
tecnología 15%

Entrega No.2: (SAD +
.NET) 10% +
Implementación (15%)
25%

REGLAS

- Netiqueta en la comunicación
- Horario de clase Martes y Jueves de 4 pm a 6 pm,
 - (horario de interacción) Lunes a viernes de 8 am a 5 pm
 - Para las clases y laboratorios 15 minutos al inicio y al final, para limpieza del espacio (medida de bioseguridad)
- Trato cordial
- Control de asistencia
- Cumplimiento en las fechas y medios de entrega
- Trabajo en equipo es diferente a división de trabajo
- Copia y/o plagio
- Bioseguridad
 - Uso de tapabocas en el salón y el laboratorio
 - No consumir alimentos en salón y laboratorio
 - Si tiene síntomas gripales, no entrar a clase o laboratorio, informar a dirección del programa y al docente.
 - Si hay condiciones de salud, se hará uso de la modalidad remota en Teams



MEDIOS DE COMUNICACIÓN

- Clase presencial
- Teams (presentaciones y material, videos - 15 días)
- Campus Virtual/BrightSpace (entregas y material)
- Correo Institucional (comunicación oficial)
- Grupo en Telegram (comunicación no oficial)





BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

CAMPUS VIRTUAL



Andres Sanchez Martin

Inicio Campus Virtual Ayuda ▾



Mis cursos ▾

Calendario ▾

<https://campusvirtuallms.javeriana.edu.co>

MOTIVACIÓN

Toda solución de software cuenta con una arquitectura

- Organiza el trabajo en equipo
- Usa estándares y lenguajes (UML)
- Identifica necesidades de los usuarios
- Modela el mundo real
- Identificar un problema y plantear una solución

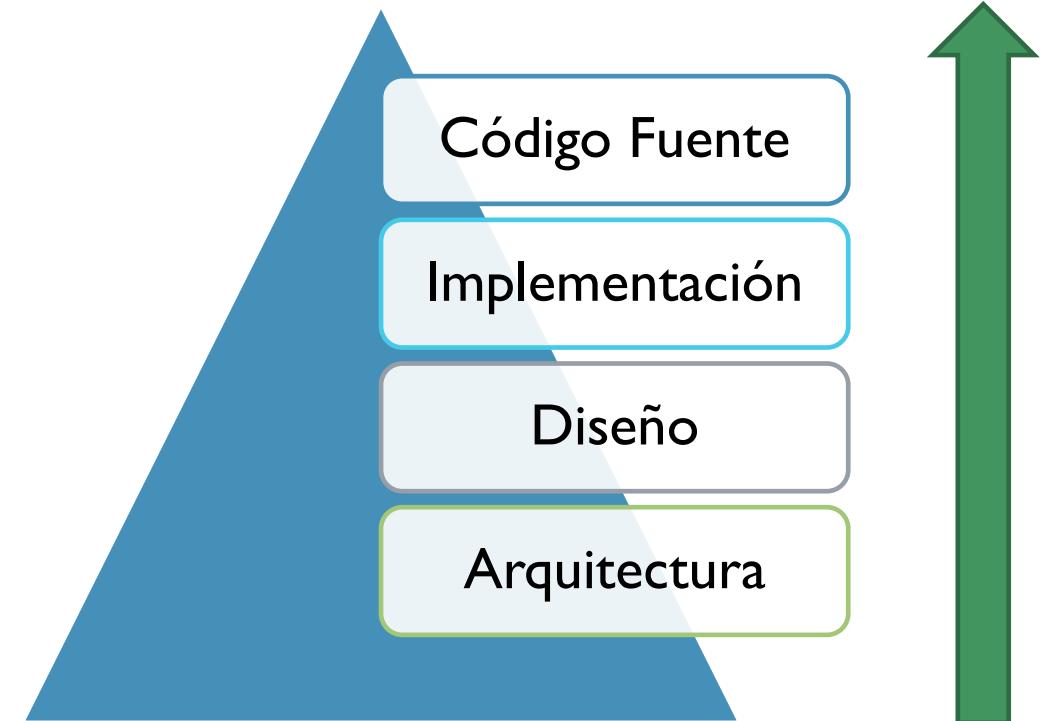


IMPORTANCIA

La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema.



- **Requerimientos**
- **Diseño**
- **Documentación**
- **Evaluación**



¿QUIÉNES SON LOS ESTUDIANTES?

- ¿y ustedes quienes son?
 - ¿Trabajo?
 - ¿Semestre?
 - ¿intereses?
 - ¿Materias que han visto?
 - ¿Expectativas del curso?





Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué es la Ingeniería de Software?
- ¿Cuál es el Proceso de Desarrollo de Software?
- ¿Qué es Arquitectura de Software?
- ¿Cuáles son los principios SOLID?
- ¿Cuáles son las características de POO?





(1306)

ARQUITECTURA DE SOFTWARE

I.2 Definiciones I

Agenda

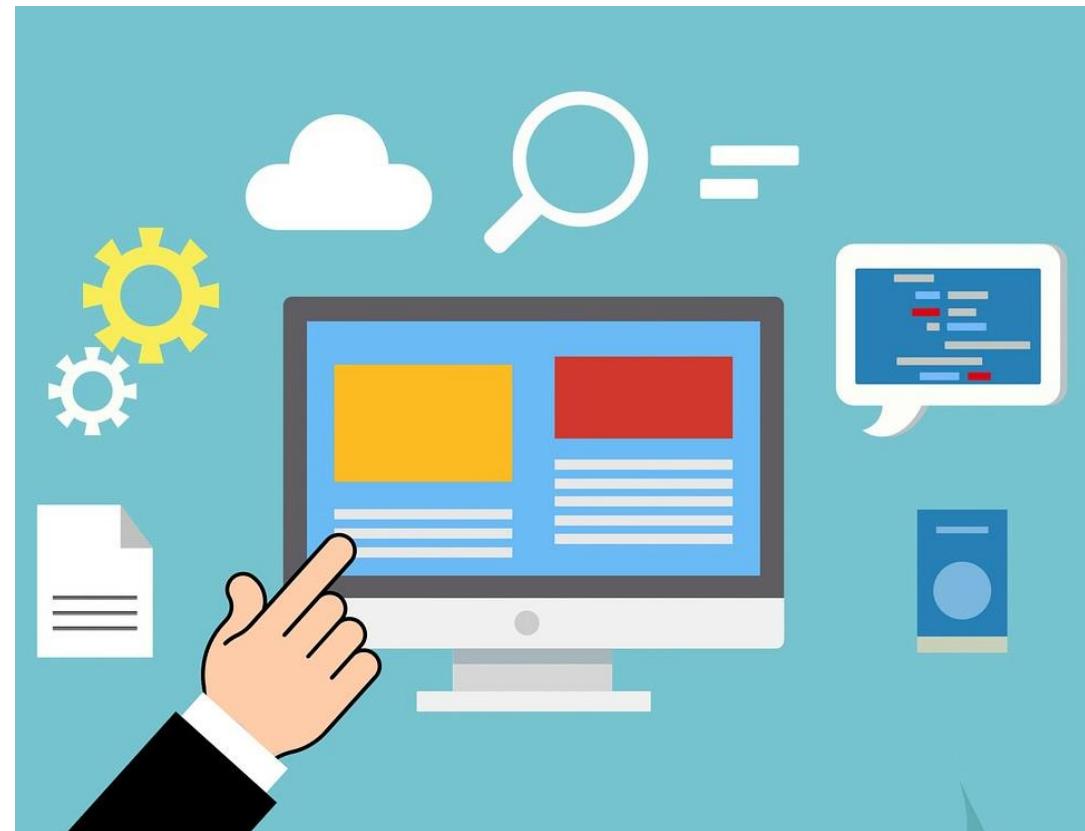
- Ingeniería de Software
- Proceso de Desarrollo de Software
- Fundamentos de arquitectura
- Arquitectura de Software
- Principios SOLID y Características POO



INGENIERÍA DE SOFTWARE

El concepto “ingeniería de software” se propuso originalmente en 1968, en una conferencia realizada para discutir lo que entonces se llamaba la “crisis del software” (Naur y Randell, 1969). Se volvió claro que los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software. Éstos no eran confiables, costaban más de lo esperado y se distribuían con demora.

A lo largo de las décadas de 1970 y 1980 se desarrolló una variedad de nuevas técnicas y métodos de ingeniería de software, tales como la programación estructurada, el encubrimiento de información y el desarrollo orientado a objetos. Se perfeccionaron herramientas y notaciones estándar y ahora se usan de manera extensa.



INGENIERÍA DE SOFTWARE: : DESARROLLO DE SOFTWARE

Es el proceso metodológico para la construcción de soluciones de software.

Existen varios enfoques (paradigmas) para desarrollar software:

- Modelo de cascada
- Modelo de espiral
- Desarrollo iterativo e incremental
- Desarrollo ágil





DESARROLLO DE SOFTWARE: DIFICULTADES

ESENCIALES

Especificación,
diseño y
comprobación del
concepto

ACCIDENTALES

Detalles de la
implementación y
producción actual

“Consideró a la especificación, diseño y comprobación del *concepto* la parte difícil de hacer software. (...)

Si esto es cierto, hacer software siempre será difícil. No existe la bala de plata.”

(Frederick P. Brooks Jr., 1986)

DESARROLLO DE SOFTWARE: DIFICULTADES/RESTOS

Andrés Armando Sánchez Martín



DESARROLLO DE SOFTWARE: SOLUCIONES

Complejidad:
No desarrollar: Comprar
- OSS

Conformidad:
Prototipado rápido

Dificultades
Esenciales

Tolerancia al cambio;
Desarrollo evolutivo

Invisibilidad:
Grandes Diseñadores
¡Arquitectos!

PROCESO DE DESARROLLO DE SOFTWARE: FASES Y ARTEFACTOS

Mejora del sistema

Disparador



Necesidad del Usuario

Descubrimiento

Análisis de Requerimientos

Análisis y documentación

Comprensión del problema

Abstracción

Diseño de la Solución

Modelado



Pontificia Universidad
JAVERIANA
Bogotá

PROCESO DE DESARROLLO DE SOFTWARE: FASES Y ARTEFACTOS

Andrés Armando Sánchez Martín

Diseño de
la Solución

Solución
detallada

Programación e
integración

Desarrollo y
Evaluación

Automatización de
pruebas

Artefacto de
Software

Infraestructura

Despliegue

Operaciones



PROCESO DE DESARROLLO DE SOFTWARE: FASES Y ARTEFACTOS

Andrés Armando Sánchez Martín

Despliegue

*Software
disponible*

Arreglo de errores

Mantenimiento y
Evolución

Nuevas funcionalidades

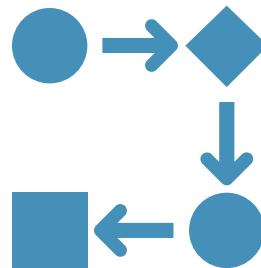
Sistema
deprecado



PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES

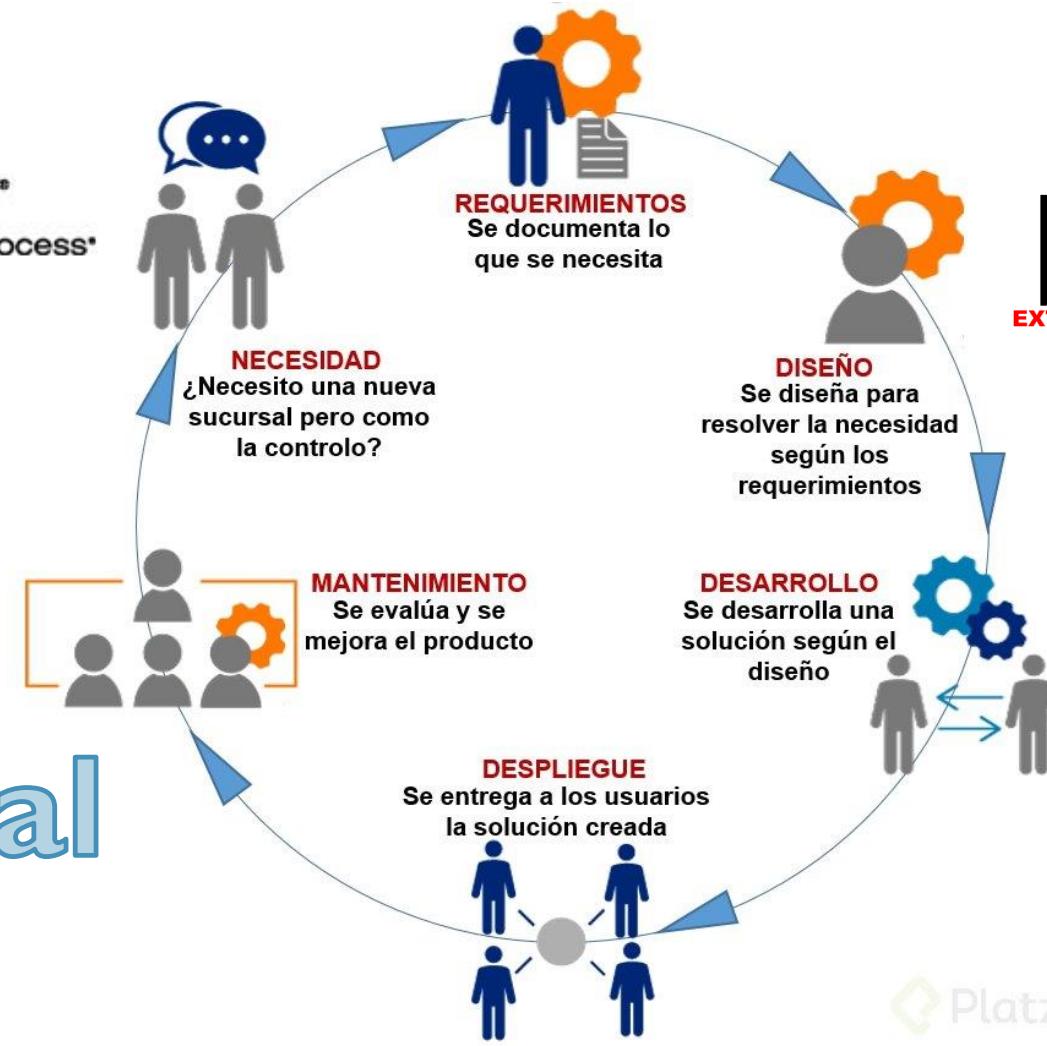
Andrés Armando Sánchez Martín

PSP
Personal Software Process



Tradicional

RUP®
Rational Unified Process®



XP
EXTREME PROGRAMMING



Ágil

PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES



Contenido

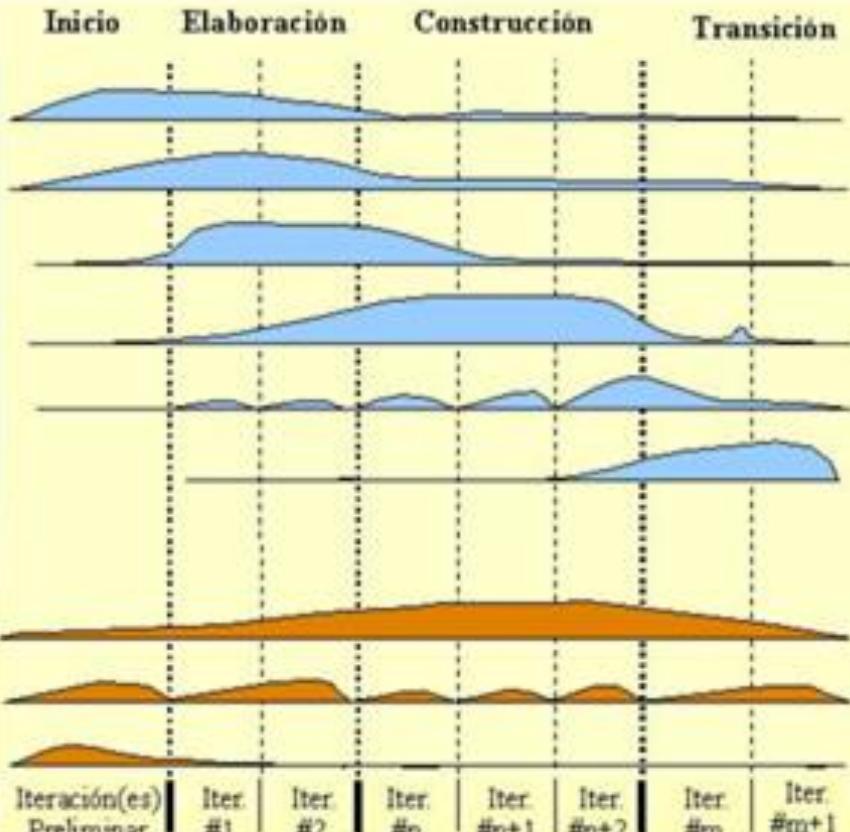
Flujos de Trabajo de Procesos

Modelación de Negocios
Requerimientos
Análisis y Diseño
Implementación
Prueba
Desarrollo

Flujos de Trabajo de Soporte

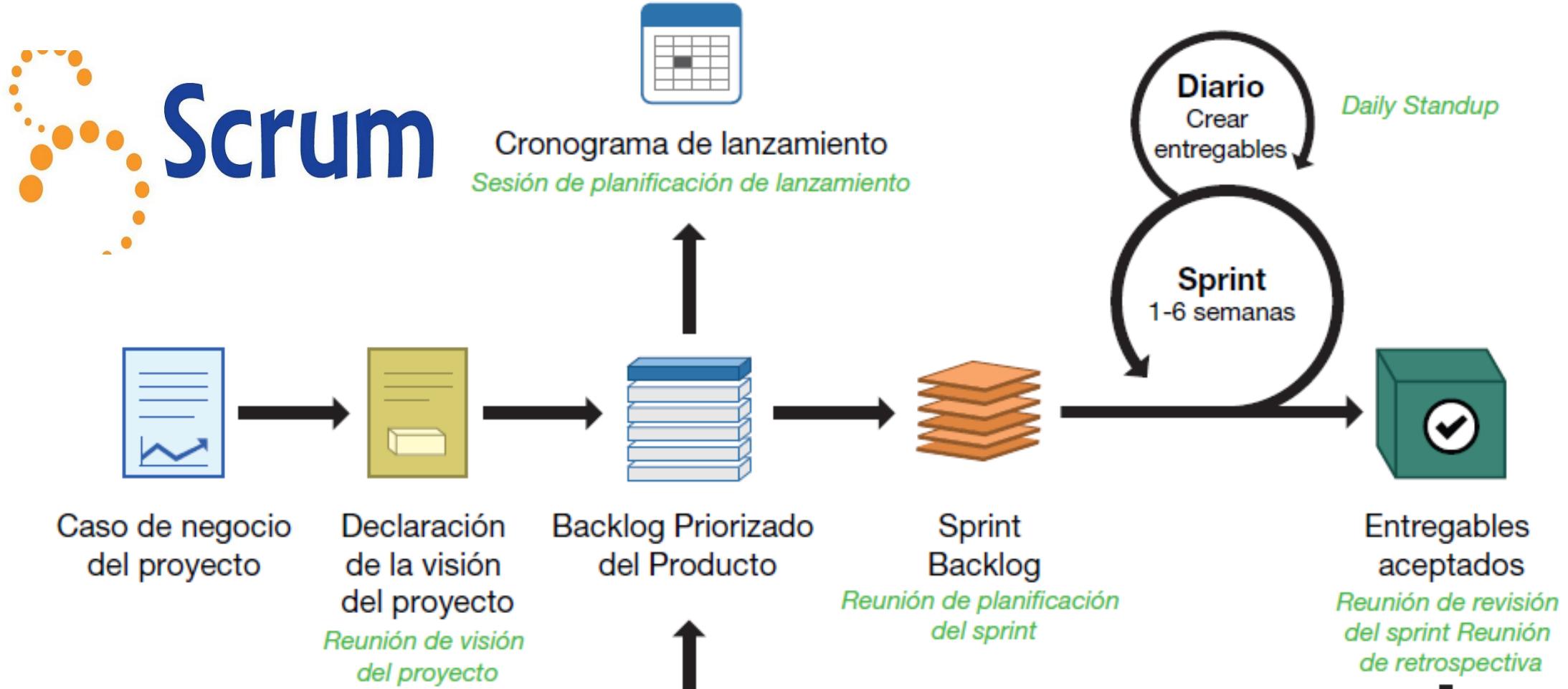
Admin. Configuración
Administración
Ambiente

Fases



Iteraciones

PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES



PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES

Andrés Armando Sánchez Martín



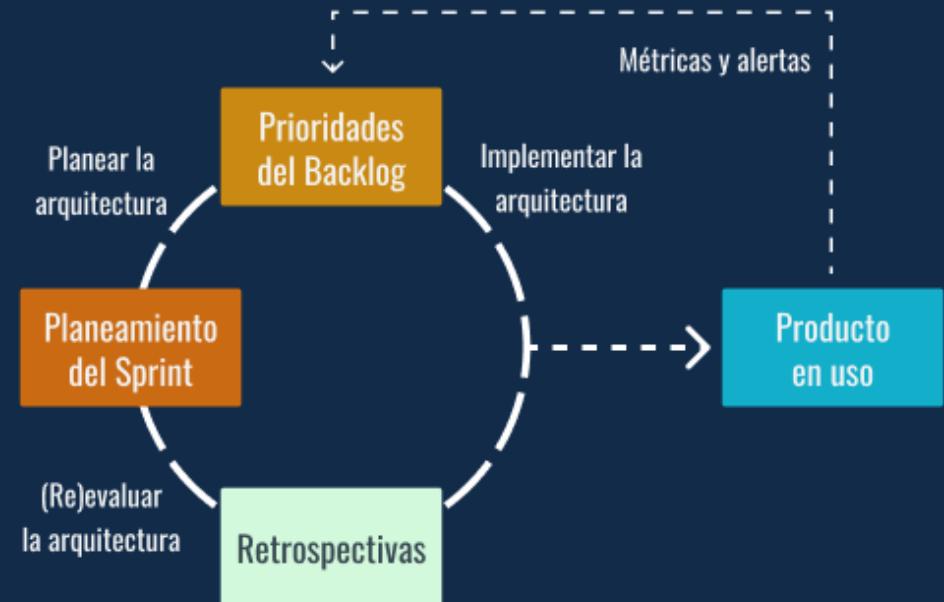
PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES

Metodologías tradicionales

Etapa de diseño



Metodologías ágiles



PROCESO DE DESARROLLO DE SOFTWARE: METODOLOGÍAS TRADICIONALES VS AGILES

- Arquitectos en la torre de marfil
 - Falta de comunicación
- Centralización de todas las decisiones
 - Arquitecto = cuello de botella
- Tomar demasiadas decisiones
 - Retrasar ciertas decisiones puede ser mejor que deshacerlas
- Big Design Up-Front
 - Demasiados diagramas y documentos innecesarios
 - Retardos debidos al proceso de arquitectura
- Arquitectura que puede reaccionar a cambios en entorno
 - Adaptación a cambios continuos
 - También conocidas como arquitecturas evolutivas
 - Buena arquitectura permite agilidad
 - Mejor comprensión de decisiones y soluciones de compromiso
- Anti-patrón común: adopción de técnicas de desarrollo de software ágil que crean arquitecturas que no son ágiles
 - Causado por demasiado enfoque en funcionalidad



PROCESO DE DESARROLLO DE SOFTWARE: ROLES

Tradicional

Experto del dominio

Analista

Administrador de sistemas (SysAdmin)

Arquitecto, Desarrollador, QA – Tester

Gestor del proyecto

Ágil

Partes interesadas (*stakeholders*)

Cliente / Dueño del producto

DevOps / SRE (*site reliability engineer*)

Equipo de Desarrollo (Team)

Facilitador

FUNDAMENTOS DE ARQUITECTURA

Architecture = Proceso y producto de planificar, diseñar y construir edificios u otras estructuras

Utilitas (utilidad):

- Ser útil y funcionar bien para las personas que lo van a usar

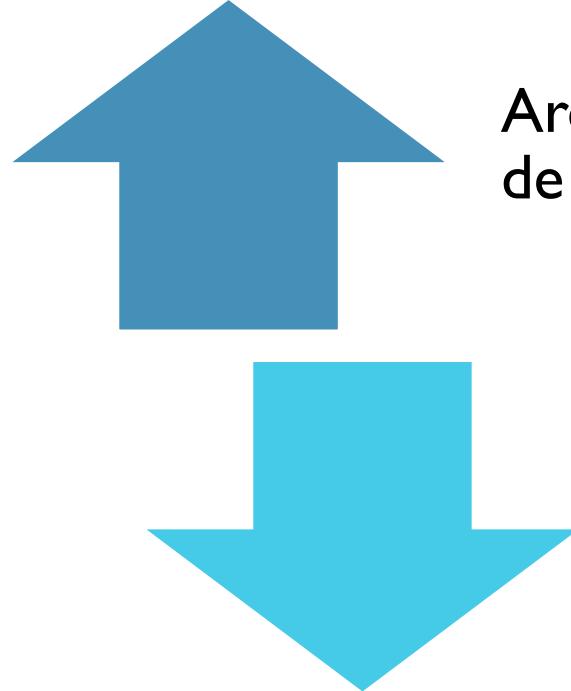
Firmitas (durabilidad):

- Mantenerse de forma robusta y en buena condición

Venustas (elegancia):

- Ser agradable a las personas

FUNDAMENTOS DE ARQUITECTURA: ARQUITECTURA VS DISEÑO



Arquitectura se enfoca en estructura
de alto nivel de un sistema de software

Decisiones de diseño principales del
sistema
Si hay que cambiarlas ⇒ Coste elevado

"Toda arquitectura conlleva diseño pero no todo el diseño es arquitectura" G. Booch



Pontificia Universidad
JAVERIANA
Bogotá

FUNDAMENTOS DE ARQUITECTURA

Andrés Armando Sánchez Martín



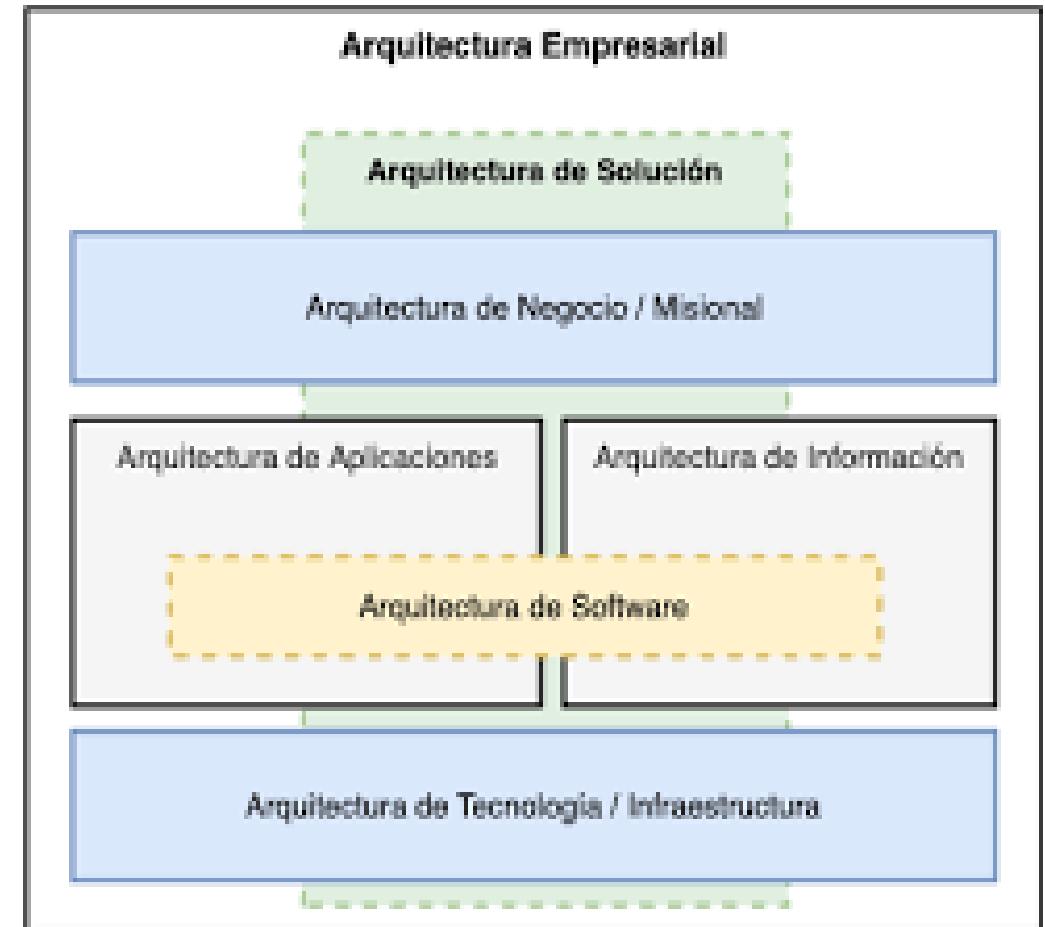
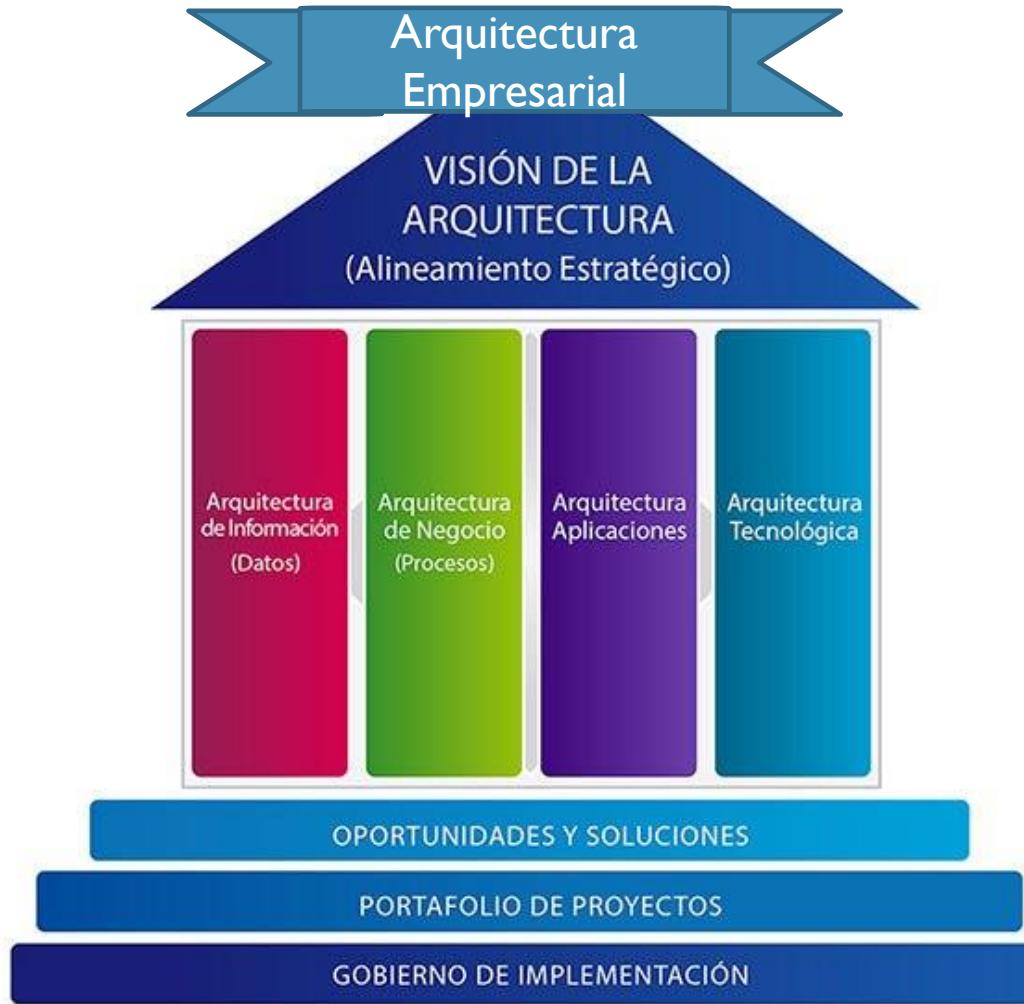
Niveles del Rol

Arquitectura
Empresarial

Arquitectura de
Sistemas

Arquitectura de
Software

FUNDAMENTOS DE ARQUITECTURA





FUNDAMENTOS DE ARQUITECTURA: LA IMPORTANCIA DE LA COMUNICACIÓN - LEY DE CONWAY

“LAS ORGANIZACIONES QUE DISEÑAN SISTEMAS ESTÁN LIMITADAS A PRODUCIR DISEÑOS QUE SON COPIAS DE LAS ESTRUCTURAS DE COMUNICACIÓN DE ESTAS ORGANIZACIONES”

Ley de Conway (Melvin Conway, 1967)

LA IMPORTANCIA DE LA COMUNICACIÓN

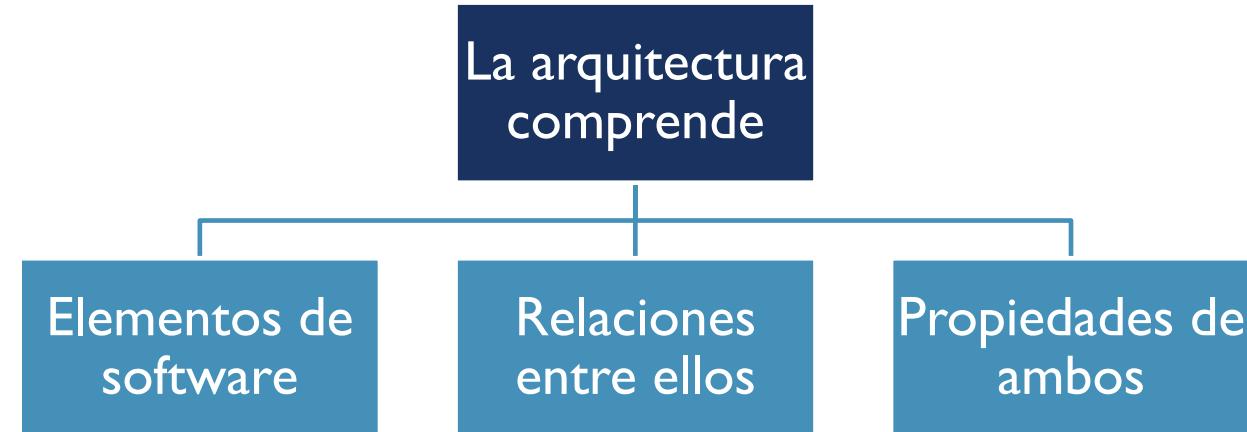
LEY DE CONWAY

Andrés Armando Sánchez Martín



ARQUITECTURA DE SOFTWARE

- Arquitectura [ISO/IEC/IEEE 42010:2011, 3.2]: Conceptos fundamentales o propiedades de un Sistema en su entorno representados por sus elementos, relaciones y los principios de su diseño y evolución
- Descripción de arquitectura: Producto de trabajo explícito que expresa una arquitectura de un sistema, normalmente a través de modelos, texto o gráficos
- Diseñar una arquitectura (architecting): Proceso de crear una arquitectura

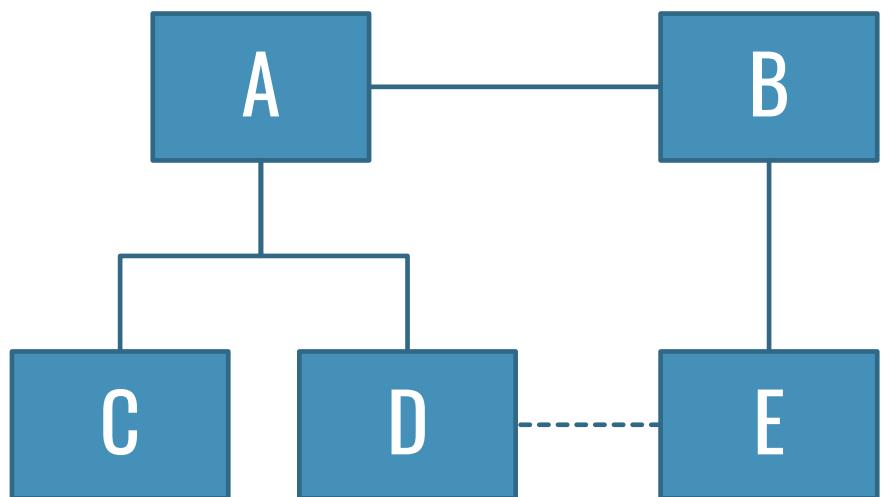


ARQUITECTURA DE SOFTWARE

- “La estructura del sistema, compuesta por elementos de software, sus propiedades visibles y sus relaciones” – Software Architecture in Practice (Bass, Clements & Kazman, 2003)
- “El conjunto de decisiones principales de diseño tomadas para el sistema” – Software Architecture: Foundations, Theory and Practice (Taylor, 2010)
- (...) la arquitectura se reduce a las cosas importantes, cualesquiera que sean” – Patterns of Enterprise Application Architecture (Fowler, 2002)

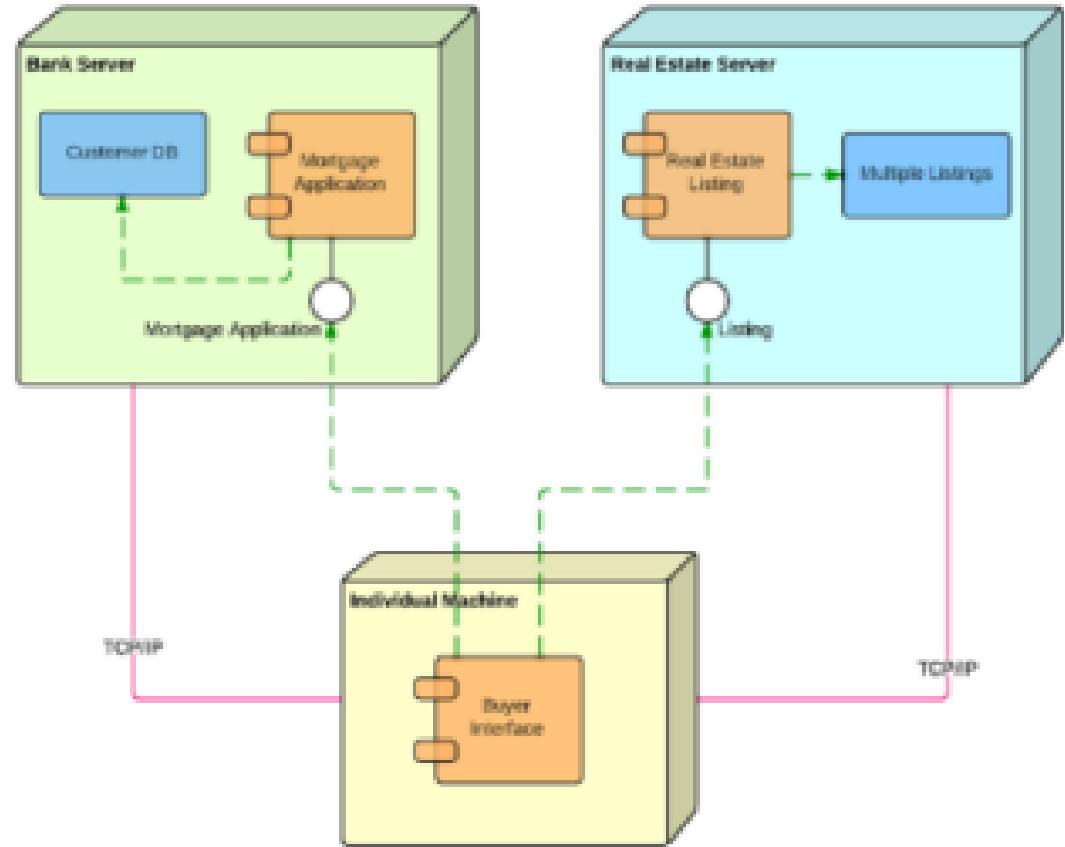
Podemos decir que:

La arquitectura de software es el diseño conceptual de una solución, donde se muestran los componentes, funciones y estructura general de una solución. Además, de mostrar como se comunican y relacionan

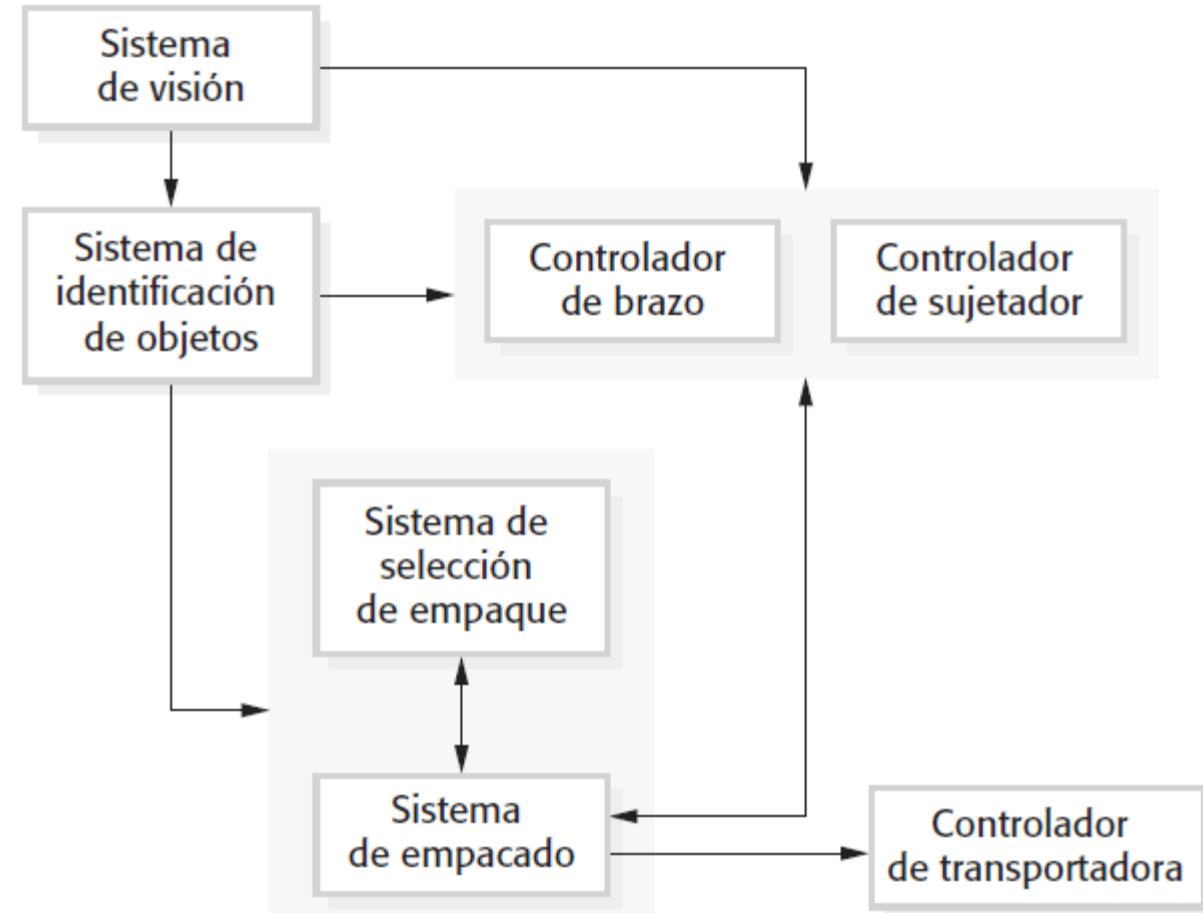


ARQUITECTURA DE SOFTWARE: CARACTERÍSTICAS

- Múltiples cambios en contexto
- Servicio/producto virtual
- No suele haber límites físicos
- Disciplina relativamente nueva
- Podemos aprender mucho de otras disciplinas

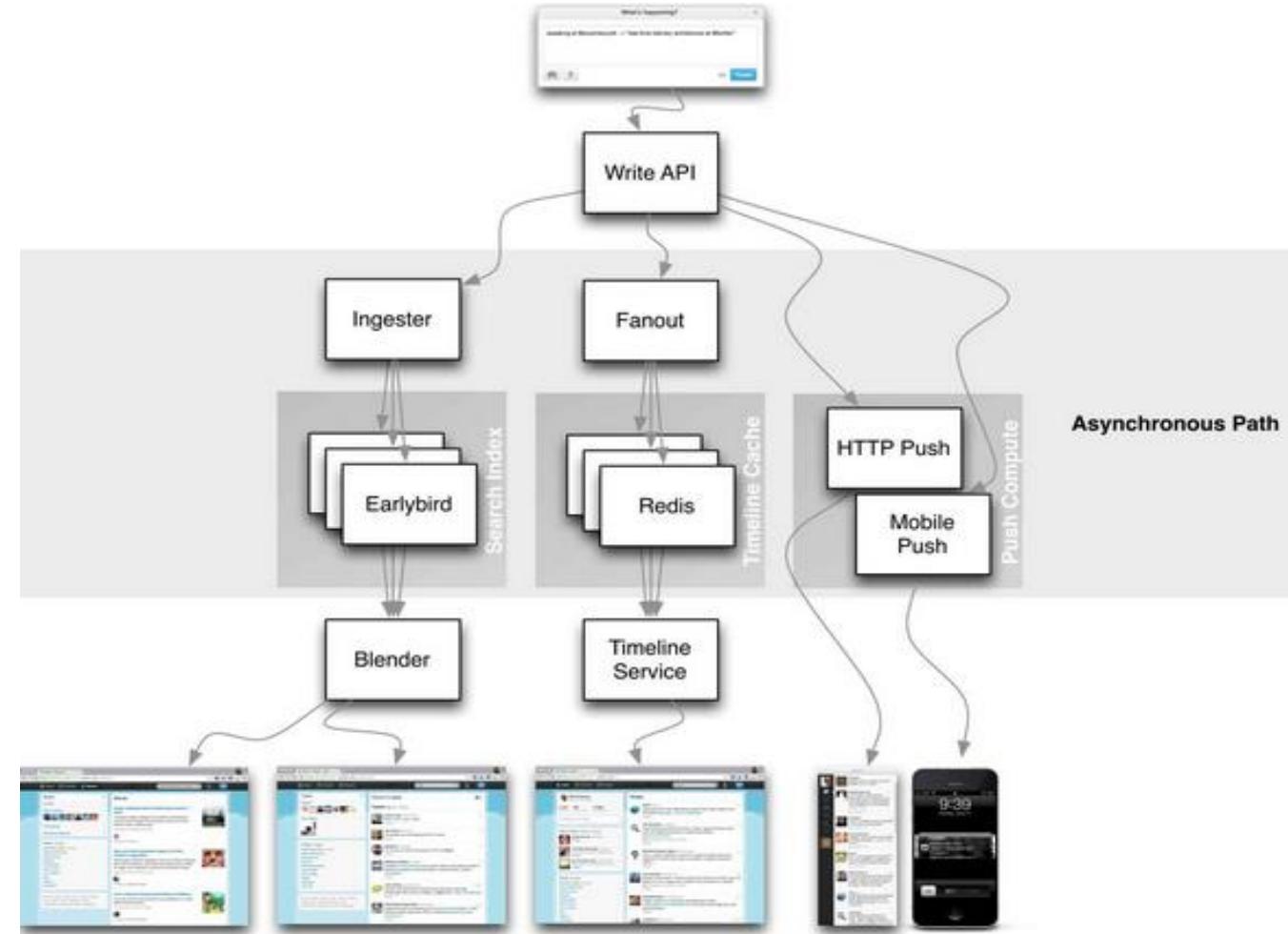


ARQUITECTURA DE SOFTWARE: EJEMPLOS



ARQUITECTURA DE SOFTWARE: EJEMPLOS

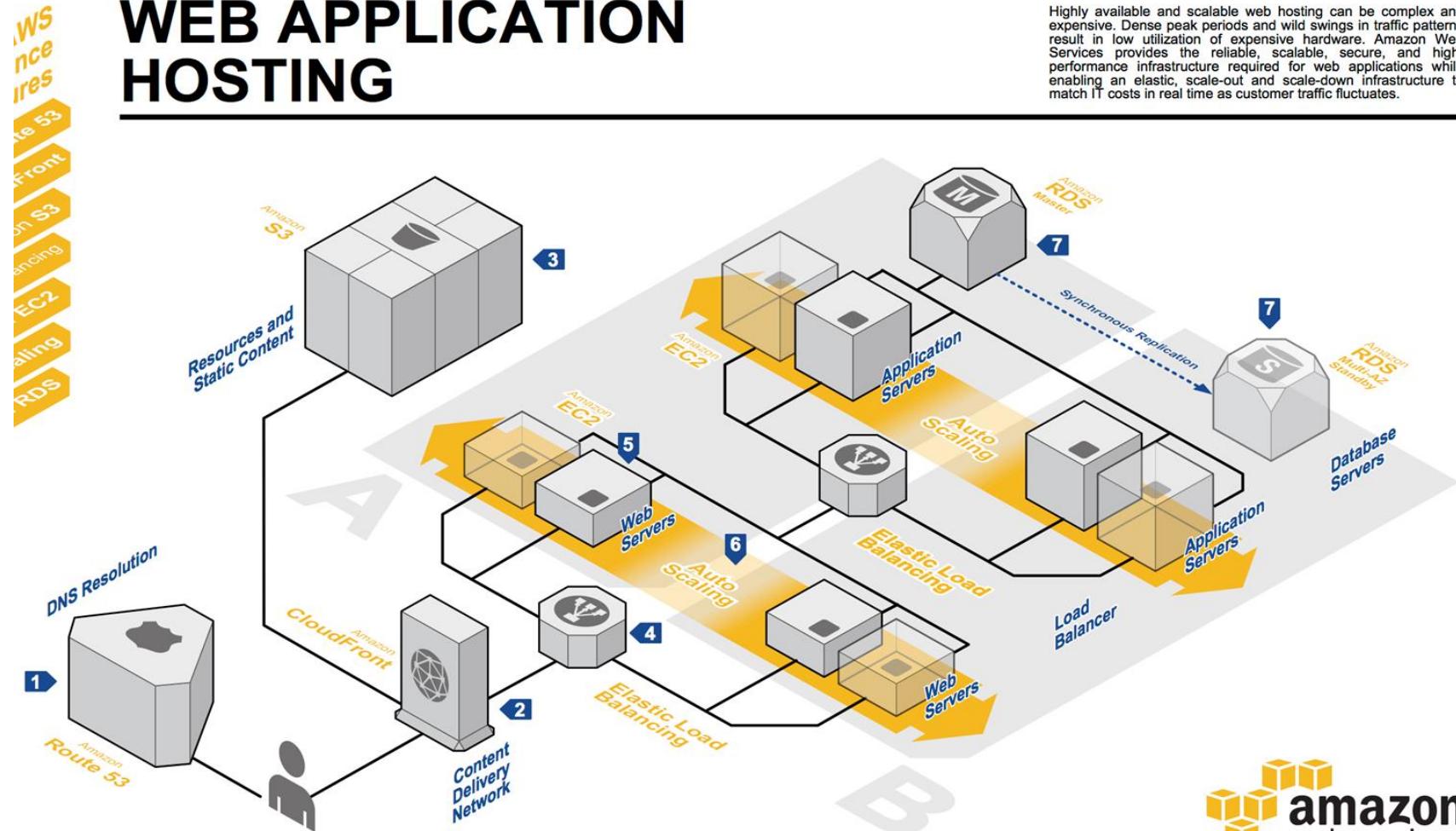
Andrés Armando Sánchez Martín



ARQUITECTURA DE SOFTWARE: EJEMPLOS

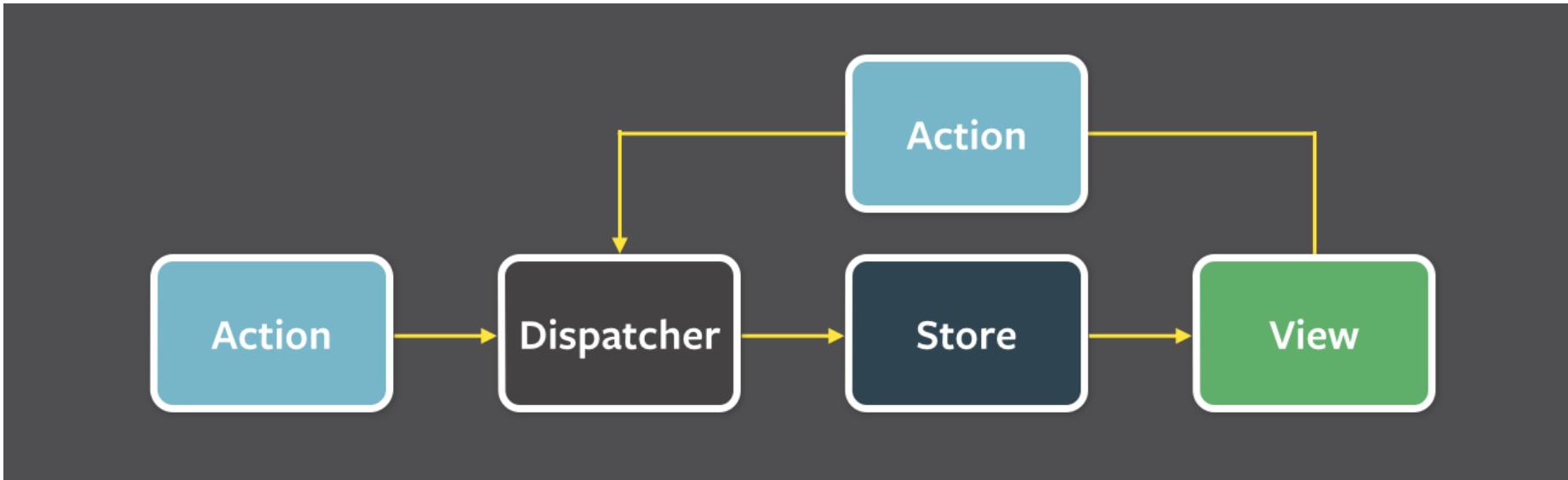
WEB APPLICATION HOSTING

Highly available and scalable web hosting can be complex and expensive. Dense peak periods and wild swings in traffic patterns result in low utilization of expensive hardware. Amazon Web Services provides the reliable, scalable, secure, and high-performance infrastructure required for web applications while enabling an elastic, scale-out and scale-down infrastructure to match IT costs in real time as customer traffic fluctuates.



ARQUITECTURA DE SOFTWARE: EJEMPLOS

Flujo de datos - Flux



ARQUITECTURA DE SOFTWARE: FUNCIONES DE UN ARQUITECTO DE SOFTWARE

ANALIZA EL CONTEXTO

Requerimientos

Funcionales

No Funcionales

Riesgos

Restricciones

POSEE HERRAMIENTAS DE DISEÑO

Atributos de Calidad

Estilos de Arquitectura

Patrones de Arquitectura

Tácticas

DISEÑA LA SOLUCIÓN

Modelo

Razonamientos

Documentación

Implementación

ARQUITECTURA DE SOFTWARE: FUNCIONES DE UN ARQUITECTO DE SOFTWARE

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



FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS

Todos los individuos, roles u organizaciones que:

- Deberían conocer la arquitectura
- Tienen que ser convencidos de la arquitectura
- Tienen que trabajar con la arquitectura o el código
- Necesitan la documentación de la arquitectura para realizar su trabajo
- Tienen que tomar decisiones sobre el sistema o su desarrollo

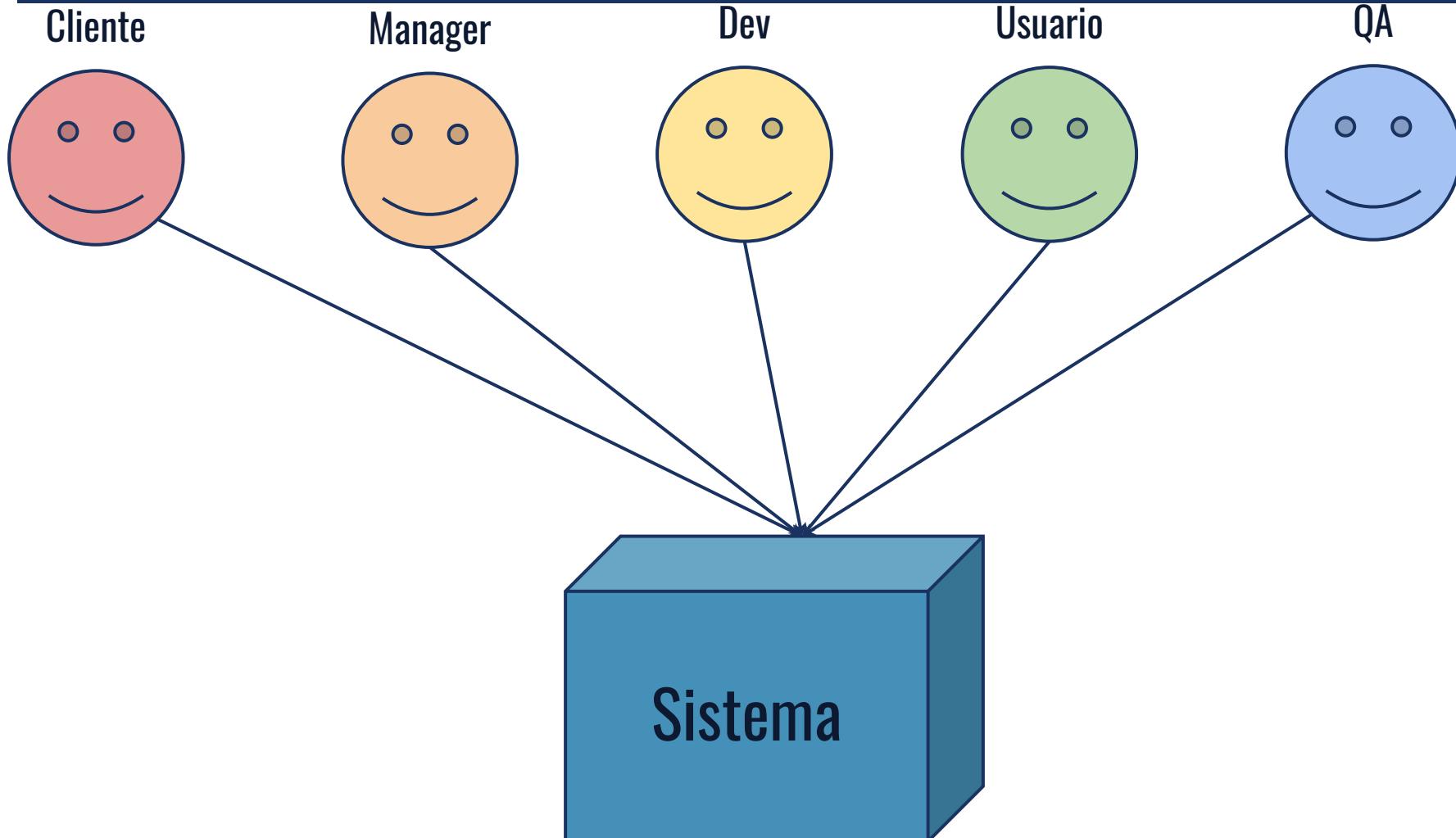
Internos

- Analista
- Diseñador
- Personas de negocios
- Desarrollador
- Product owner
- Diseñador de UX
- Jefe de proyecto
- Etc

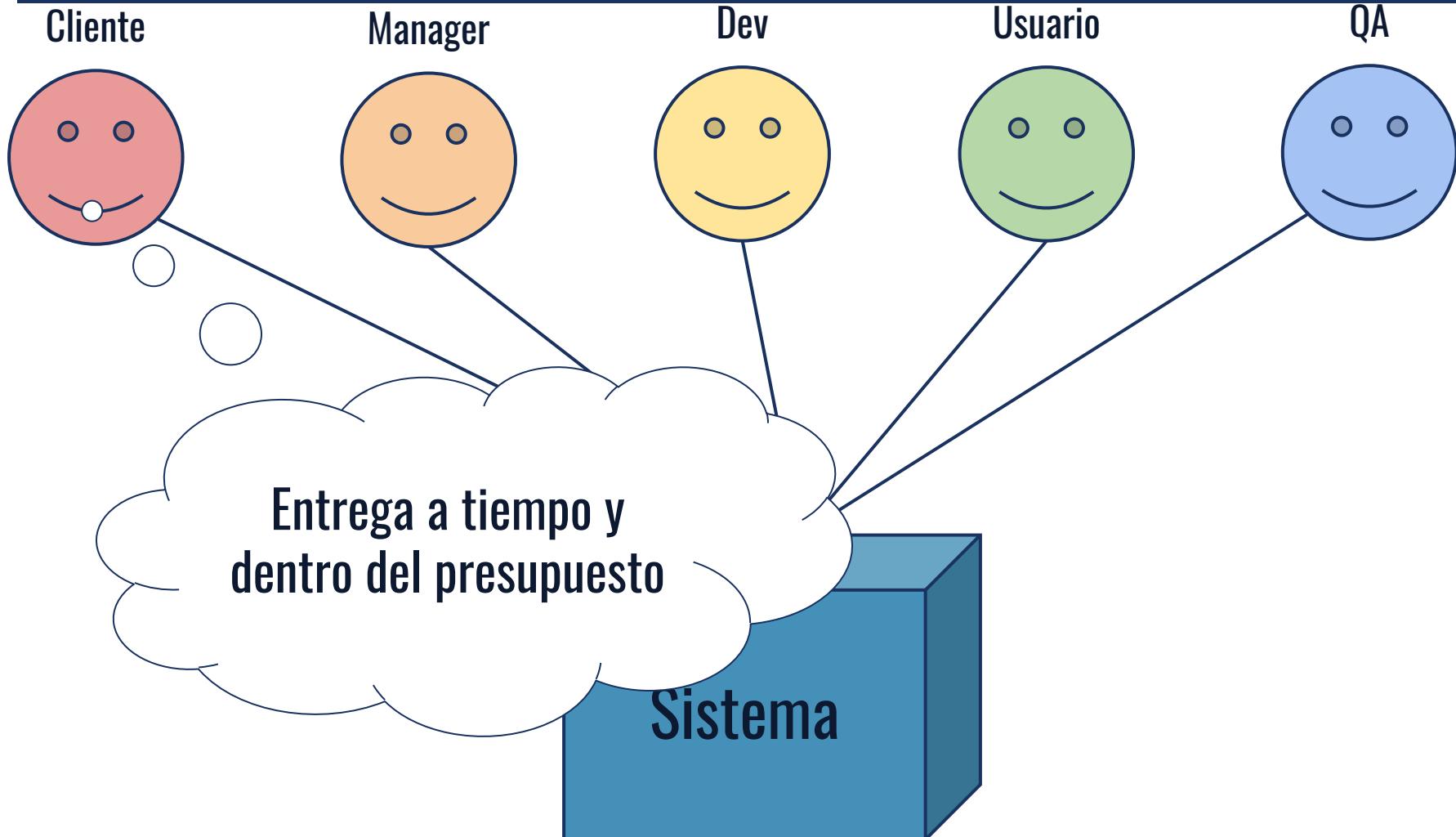
Externo

- Cliente
- Usuarios finales
- Auditor
- Autoridades públicas
- Suministradores
- Proveedores servicios externos
- Etc

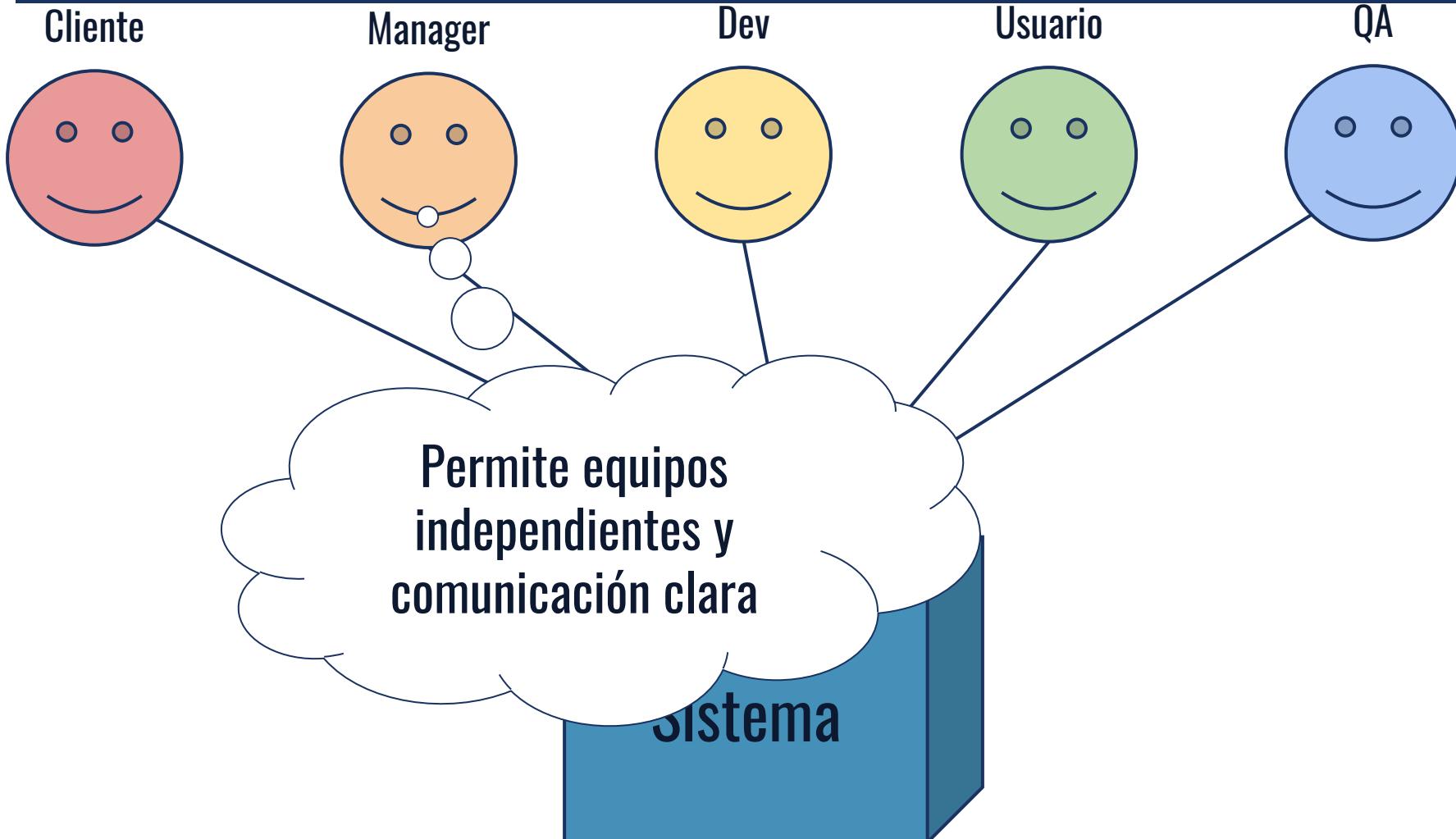
FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



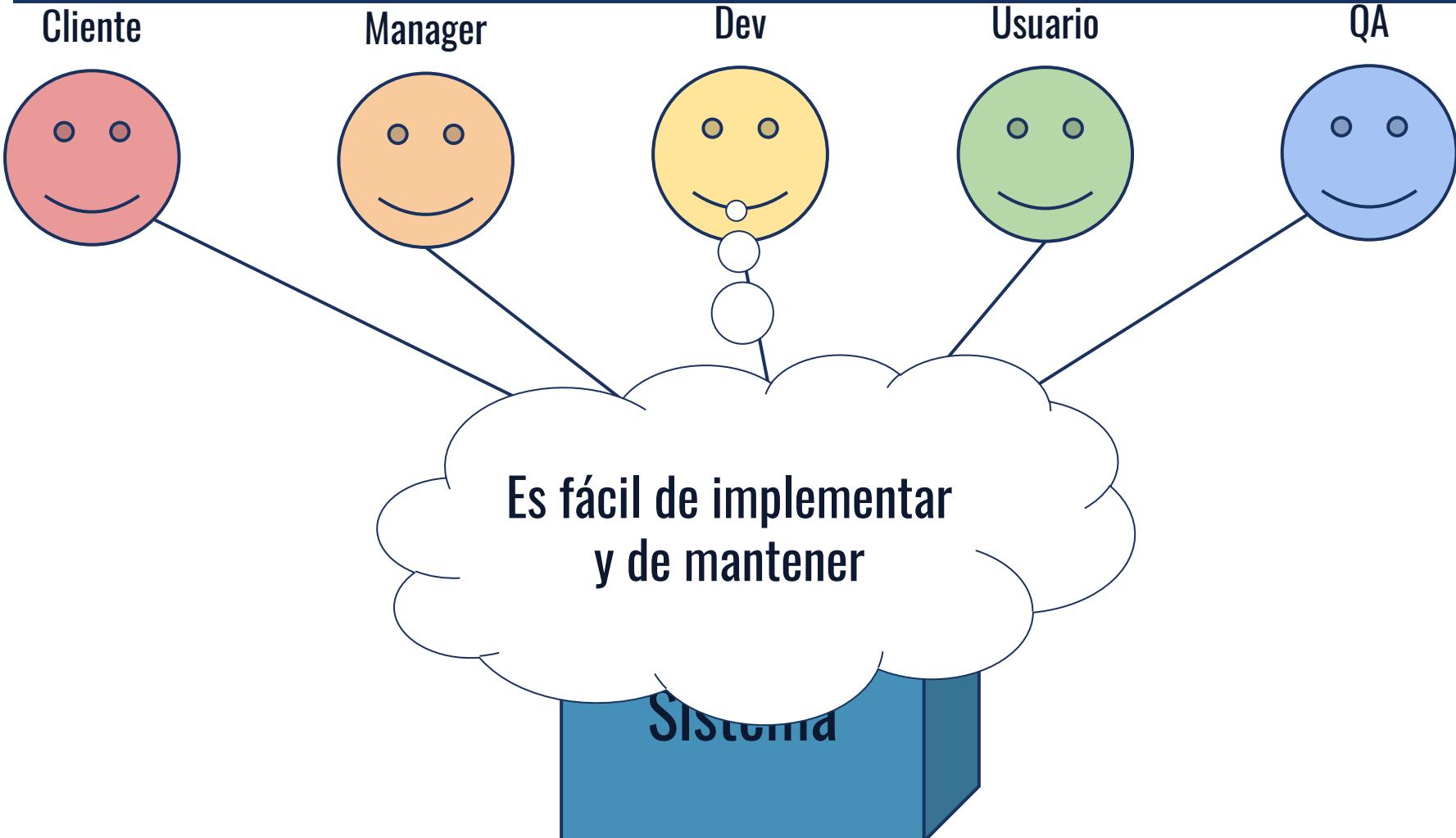
FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



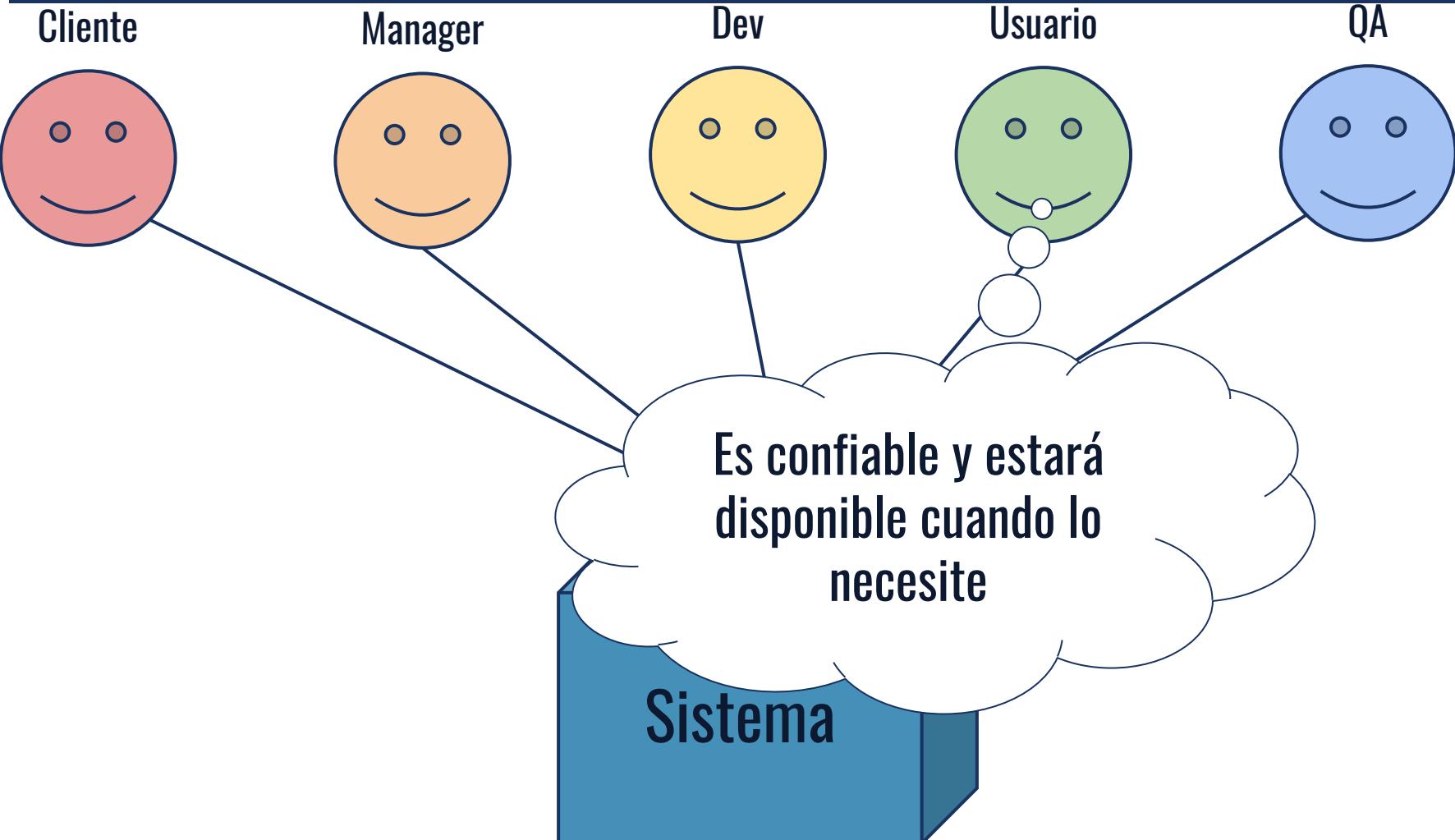
FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



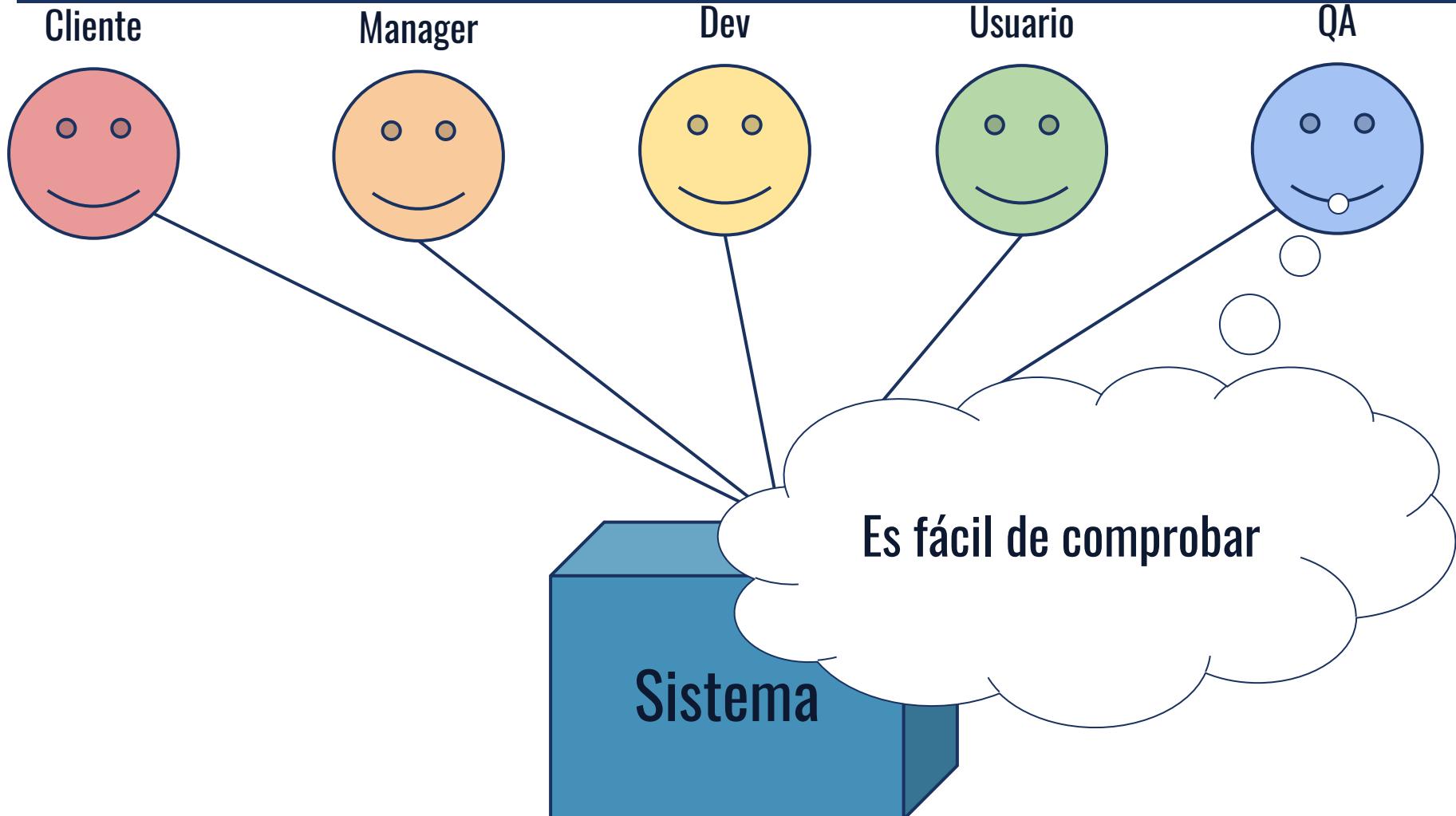
FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



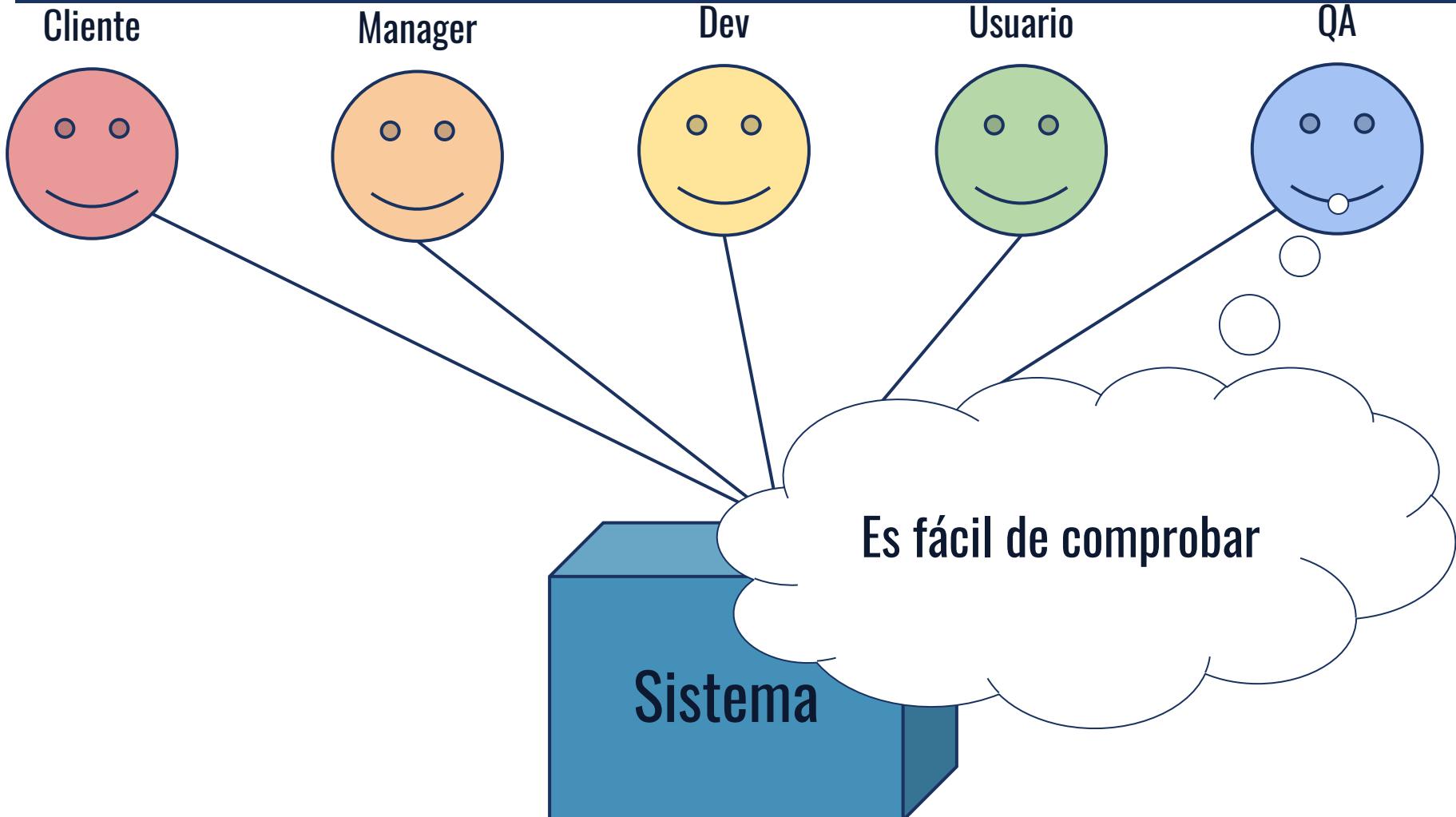
FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



FUNCIONES DE UN ARQUITECTO DE SOFTWARE: IDENTIFICAR INVOLUCRADOS



PRINCIPIOS SOLID

S – Single Responsibility Principle (SRP):

- Según este principio “una clase debería tener **una, y solo una, razón para cambiar**”. Es esto, precisamente, “razón para cambiar”, lo que Robert C. Martin identifica como “responsabilidad”.

O – Open/Closed Principle (OCP):

- Deberías ser capaz de extender el comportamiento de una clase, sin modificarla”. En otras palabras: las clases que usas deberían estar **abiertas para poder extenderse y cerradas para modificarse**.

L – Liskov Substitution Principle (LSP):

- **“las clases derivadas deben poder sustituirse por sus clases base”**. Esto significa que los objetos deben poder ser reemplazados por instancias de sus subtipos sin alterar el correcto funcionamiento del sistema o lo que es lo mismo: si en un programa utilizamos cierta clase, **deberíamos poder usar cualquiera de sus subclases** sin interferir en la funcionalidad del programa.

PRINCIPIOS SOLID

I – Interface Segregation Principle (ISP):

- “Haz interfaces que sean específicas para un tipo de cliente”, es decir, para **una finalidad concreta**.

D – Dependency Inversion Principle (DIP)

- “**Depende de abstracciones**, no de clases concretas”. 1. Los módulos de alto nivel **no deberían depender de módulos de bajo nivel**. Ambos deberían depender de abstracciones. 2. **Las abstracciones no deberían depender de los detalles**. Los detalles deberían depender de las abstracciones.

CARACTERÍSTICAS POO

Abstracción

Este proceso permite identificar características y comportamientos, relevantes y comunes, permitiendo así definir las entidades y como se relacionan y comunican entre ellas.

Encapsulamiento

Es una barrera protectora que evita que el código y los datos sean accedidos por azar o error por otros fragmentos de código o fuera de la clase.

Modularidad

Se basa en el principio de “Divide y vencerás”, dividir una aplicación en componentes llamados “módulos” los que son independientes al resto de la aplicación (en lo posible) reduciendo la dependencia..

CARACTERÍSTICAS POO

Principio de Ocultación

Es el proceso de ocultar todos los detalles internos de un objeto al mundo exterior.

Polimorfismo

- Es la capacidad por la cual, podemos crear funciones o variables de referencia, que se comporta de manera diferente en diferentes contexto programático

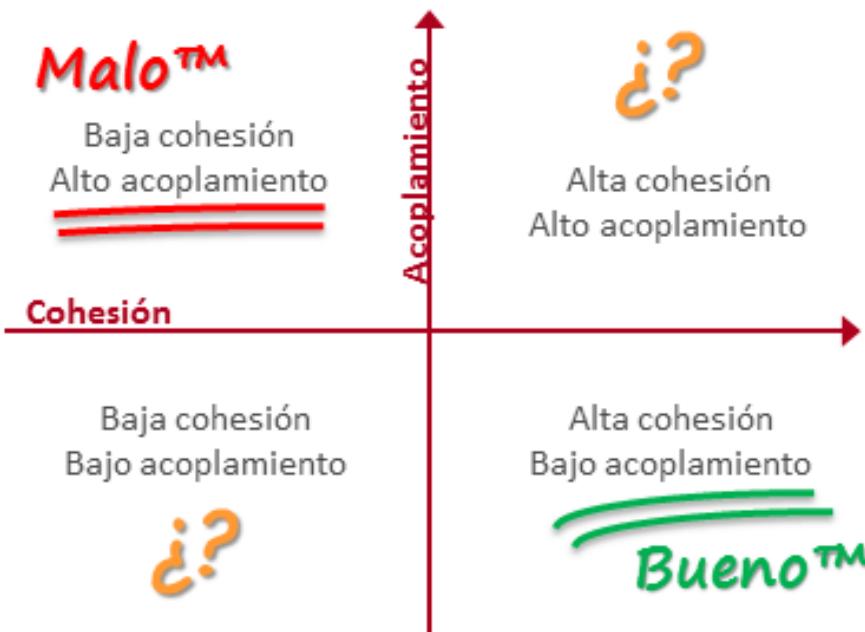
Herencia

Poder juntar las características y comportamientos comunes en entidades, y organizarlas de manera jerárquica para que se relacionen con otras.

Recolector de basura

Optimizar el uso de los recursos de hardware de una manera transparente para el usuario y programador.

SOLID Y POO: COHESIÓN Y ACOPLAMIENTO



Acoplamiento

- El acoplamiento se refiere al **grado de interdependencia que tienen dos unidades de software entre sí**, entendiendo por unidades de software: clases, subtipos, métodos, módulos, funciones, bibliotecas, etc.
- Si dos unidades de software son completamente independientes la una de la otra, decimos que están desacopladas.

Cohesión

- La cohesión de software es el **grado en que elementos diferentes de un sistema permanecen unidos para alcanzar un mejor resultado** que si trabajaran por separado. Se refiere a la forma en que podemos agrupar diversas unidades de software para crear una unidad mayor

BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué es un Patrón de Software?
- ¿Cuáles son los Estilos Arquitectónicos?
- ¿Cuáles son los Patrones Arquitectónicos?
- ¿Qué son los Requerimientos?
- ¿Qué son los modelos de calidad de Software?

Tarea:

- Hacer un cuadro comparativo de los patrones de diseño de software y los patrones de arquitectura de software
- Definición, diagrama, ventajas, desventaja y componentes
- Entrega clase 2.2





(1306)

ARQUITECTURA DE SOFTWARE

2.1 Definiciones 2

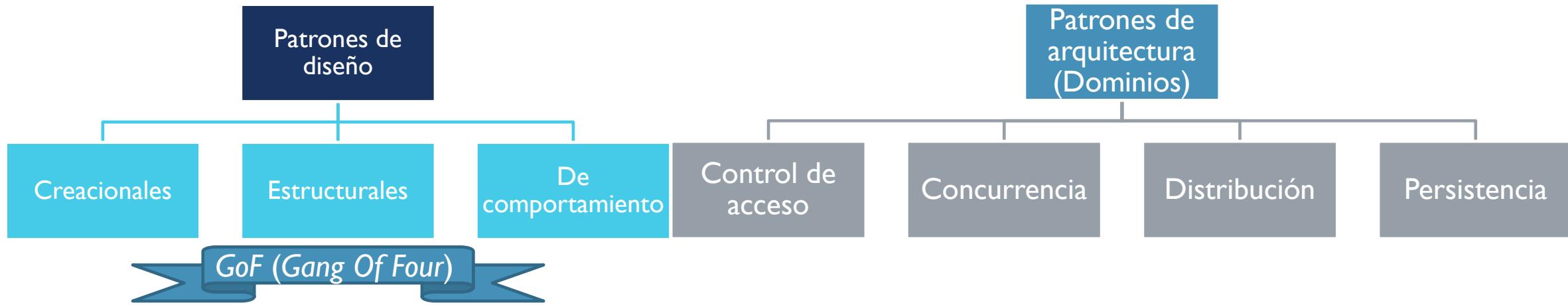
Agenda

- Patrón de Software
- Patrones Arquitectónicos
- Estilos Arquitectónicos
- Estilos y patrones Arquitecturales
- Proceso de Arquitectura
- Atributos de Calidad



PATRONES SOFTWARE

Los patrones de diseño son **modelos muestra, que sirven como guía para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos del diseño de interacción o interfaces.**



Un patrón de diseño es una manera de resolver un problema dentro de un contexto. En otras palabras, los patrones son plantillas para soluciones a problemas comunes en el desarrollo de software que se pueden usar en diferentes contextos.



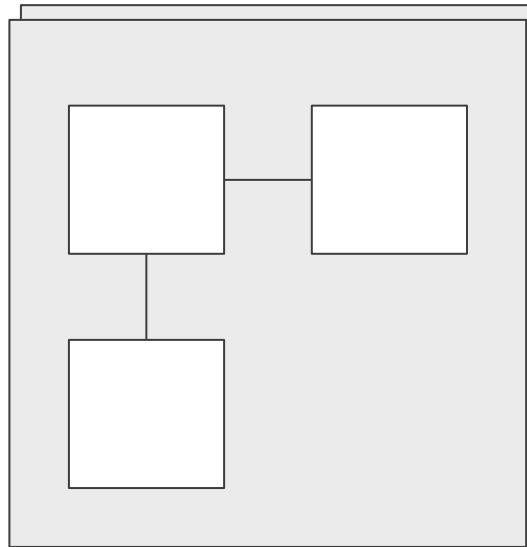
PATRONES ARQUITECTURALES

AUNQUE UN PATRÓN ARQUITECTÓNICO TRANSMITE UNA IMAGEN DE UN SISTEMA, **NO ES UNA ARQUITECTURA COMO TAL**. UN PATRÓN ARQUITECTÓNICO ES MÁS BIEN UN CONCEPTO QUE RESUELVE Y DELINEA ALGUNOS ELEMENTOS COHESIVOS ESENCIALES DE UNA ARQUITECTURA DE SOFTWARE. INNUMERABLES ARQUITECTURAS DIFERENTES PUEDEN IMPLEMENTAR EL MISMO PATRÓN Y, POR LO TANTO, COMPARTIR LAS CARACTERÍSTICAS RELACIONADAS. ADEMÁS, LOS PATRONES A MENUDO SE DEFINEN COMO ALGO "**ESTRICTAMENTE DESCrito Y COMÚNMENTE DISPONIBLE**".

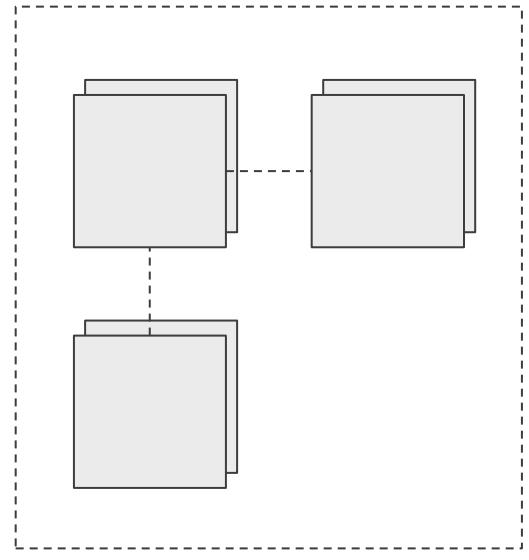
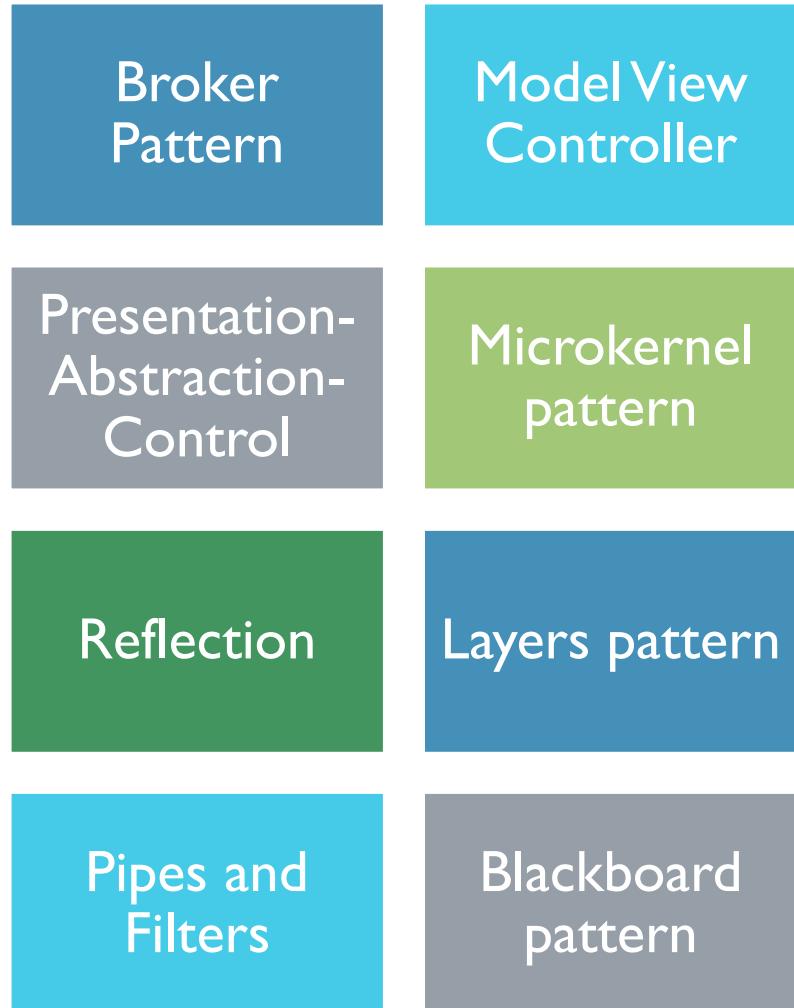
Architectural Patterns: Definition (INFOTECH, 2012)

PATRONES ARQUITECTURALES: CLASIFICACIÓN

Andrés Armando Sánchez Martín



MONOLÍTICOS



DISTRIBUIDOS



ESTILOS ARQUITECTURALES

UN ESTILO DE ARQUITECTURA ES UNA **COLECCIÓN DE DECISIONES DE DISEÑO**, APLICABLES EN UN CONTEXTO DETERMINADO, QUE **RESTRINGEN LAS DECISIONES ARQUITECTÓNICAS ESPECÍFICAS** EN ESE CONTEXTO Y OBTIENEN BENEFICIOS EN CADA SISTEMA RESULTANTE.

Software Architecture: Foundations, Theory and Practice (Taylor, 2010)

ESTILOS ARQUITECTURALES: CLASIFICACIÓN

Estilos monolíticos

Eficiencia	Curva de aprendizaje
Capacidad de Prueba	Capacidad de Modificación

un solo equipo, una sola pieza de software.

Influenciados por los Lenguajes de Programación

- Programación estructurada
- Orientado a Objetos

Flujo de Datos

- Batch
- Pipes and Filters

Capas

- Máquinas Virtuales
- Cliente Servidor
- n-Tier
- Micro Services

Memoria Compartida

- Blackboard
- Rule Based

Invocación Implícita

- Event-based
- Publisher-suscriber

Peer-to-Peer

Interprete

Estilos distribuidos

Modularidad	Disponibilidad
Uso de Recursos	Adaptabilidad

múltiples equipos independientes, múltiples sistemas intercomunicados.

ESTILOS Y PATRONES ARQUITECTURALES

- Un patrón es una solución prediseñada para un problema que puede aplicarse en cualquier contexto donde se presente el problema.
- Un estilo es un modelo conceptual que define un vocabulario común. algunas decisiones de diseño arquitectural pueden actuar al mismo tiempo como estilos y patrones.

ESTILOS Y PATRONES ARQUITECTURALES: TAREA

Blackboard

Client /
Server

Publish
subscribe

Multi tier

MVC

Web Service

Interceptores

Micro
servicios

Hexagonal

PROCESO DE ARQUITECTURA

Dominio del problema

Objetivos de
Diseño

Requerimientos
Funcionales

Atributos de
Calidad

Restricciones

Preocupaciones
(Concerns)

Riesgos

Entradas

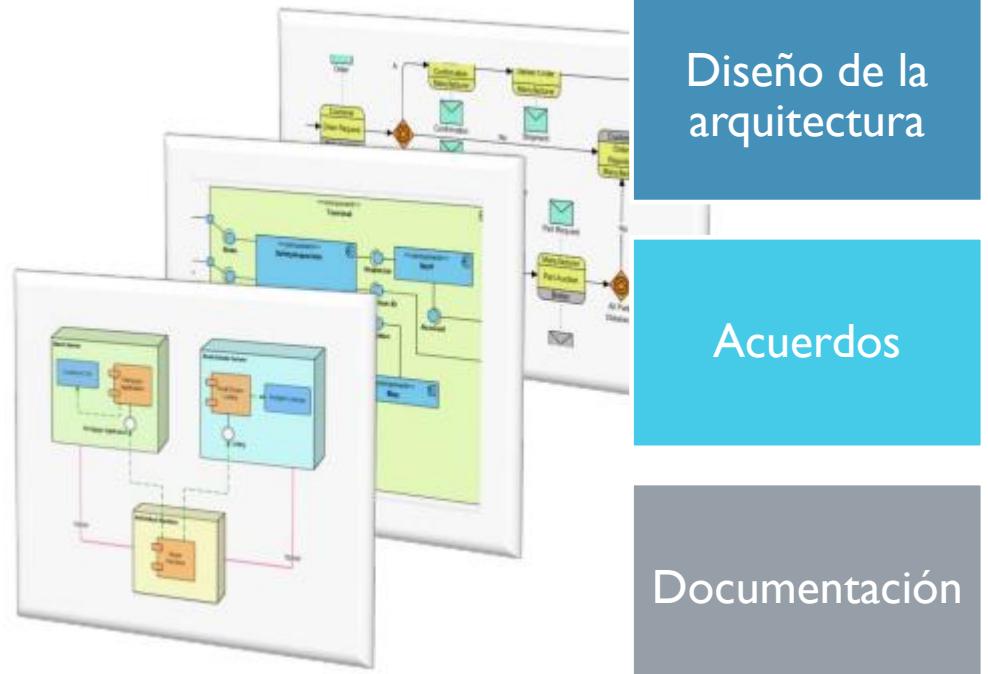


Arquitecto

Actividad de Diseño

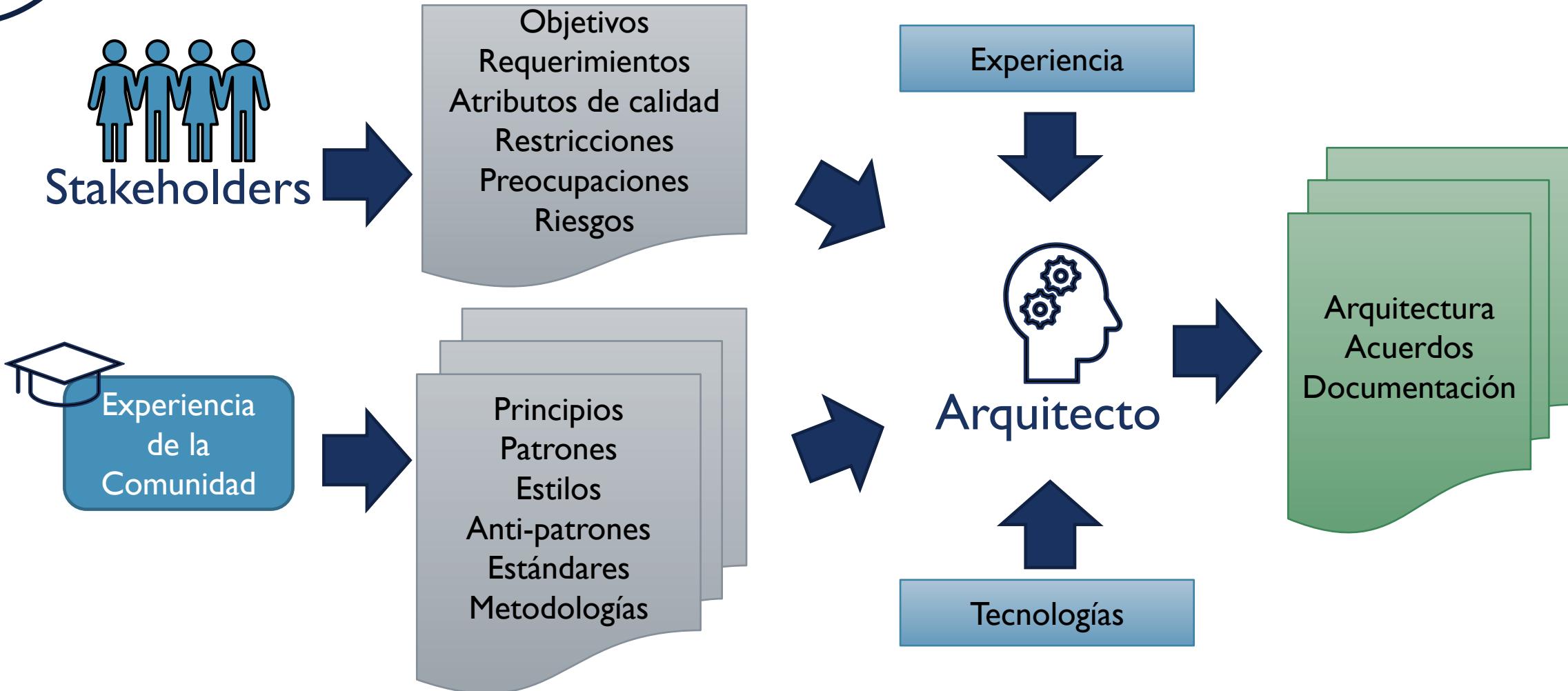
Dominio de la Solución

Diseño de la
arquitectura

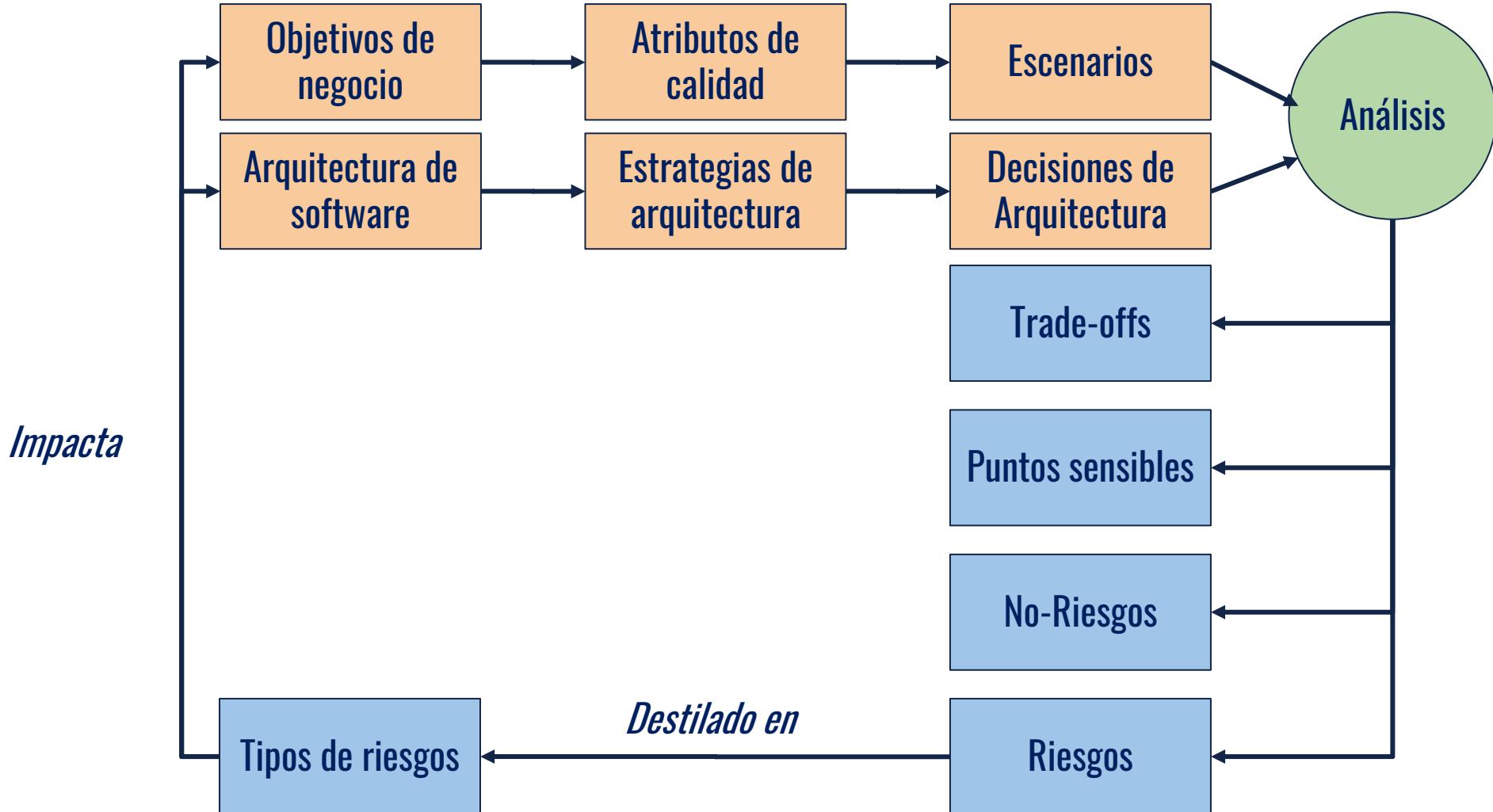


Salidas

PROCESO DE ARQUITECTURA: ARQUITECTO



PROCESO DE ARQUITECTURA

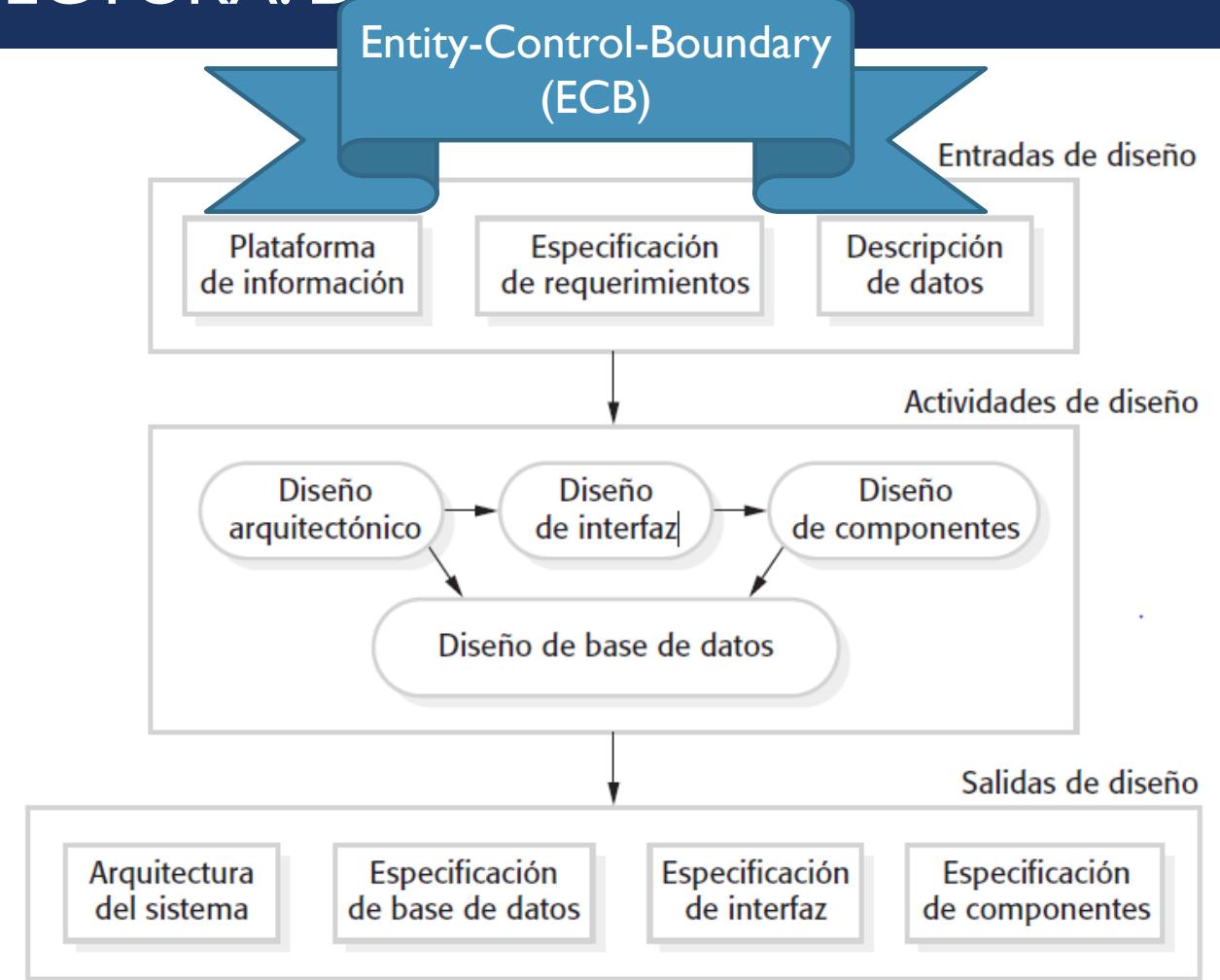


PROCESO DE ARQUITECTURA: DISEÑO

Diseño arquitectónico
(Alto nivel)

Diseño Lógico

Diseño Detallado



ATRIBUTOS DE CALIDAD

“Los atributos de calidad son las expectativas de usuario, en general implícitas, de cuán bien funcionará un producto.” - Software Requirements: 3rd Edition (Wiegers, Betty, 2013)

Performance / Rendimiento

Escalabilidad

Modificabilidad

Interoperabilidad

Seguridad

Disponibilidad

Integración

Idoneidad Funcional

Eficiencia de ejecución

Compatibilidad

Usabilidad

Confiabilidad

Seguridad

Mantenibilidad

Portabilidad

ISO/IEC
25000:2014

BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Cuáles son los tipos de requerimientos?
- ¿Cuáles son los atributos de calidad de FURPS+?
- ¿Cuáles son los atributos de calidad de ISO 25000?

Presentación:

- Explicar el estilo u patrón asignado explicando:
- Definición, historia, componentes (elementos), casos de estudio relevantes (casos de aplicación), ventajas, desventajas,
- Entrega Clase 3,2





(1306)

ARQUITECTURA DE SOFTWARE

2.2 Atributos de Calidad I

Agenda

- Requerimientos
- Modelos de Calidad de Software
- ISO 25000 vs FURPS+
- Atributos de Calidad



REQUERIMIENTOS: ENTENDER EL PROBLEMA

A la hora de definir requerimientos y analizarlos es muy importante entender el problema que estamos resolviendo

Espacio del Problema:

- Idea: ¿Qué queremos resolver?
- Criterios de éxito: ¿Cómo identificamos si estamos resolviendo el problema?
- Historias de usuario: Supuestos de historias de lo que va a ganar el usuario al utilizar la solución usando las características del problema a resolver.

Solución:

- Diseño: todo lo referente a la planificación del software, desde diseño UI, UX hasta diseño de sistemas
- Desarrollo: escribir el código, configuraciones y contrataciones de servicios
- Evaluación: medir la eficiencia y eficacia del software frente al problema
- Criterios de aceptación: medir el impacto del software, no importa lo bueno que sea el problema si los usuarios no lo usan o no le ven uso
- Despliegue (deploy): lanzar el software en ambientes productivos (mercado) y empezar a mejorar las características con un feedback loop (crear, medir, aprender)



Pontificia Universidad
JAVERIANA
Bogotá

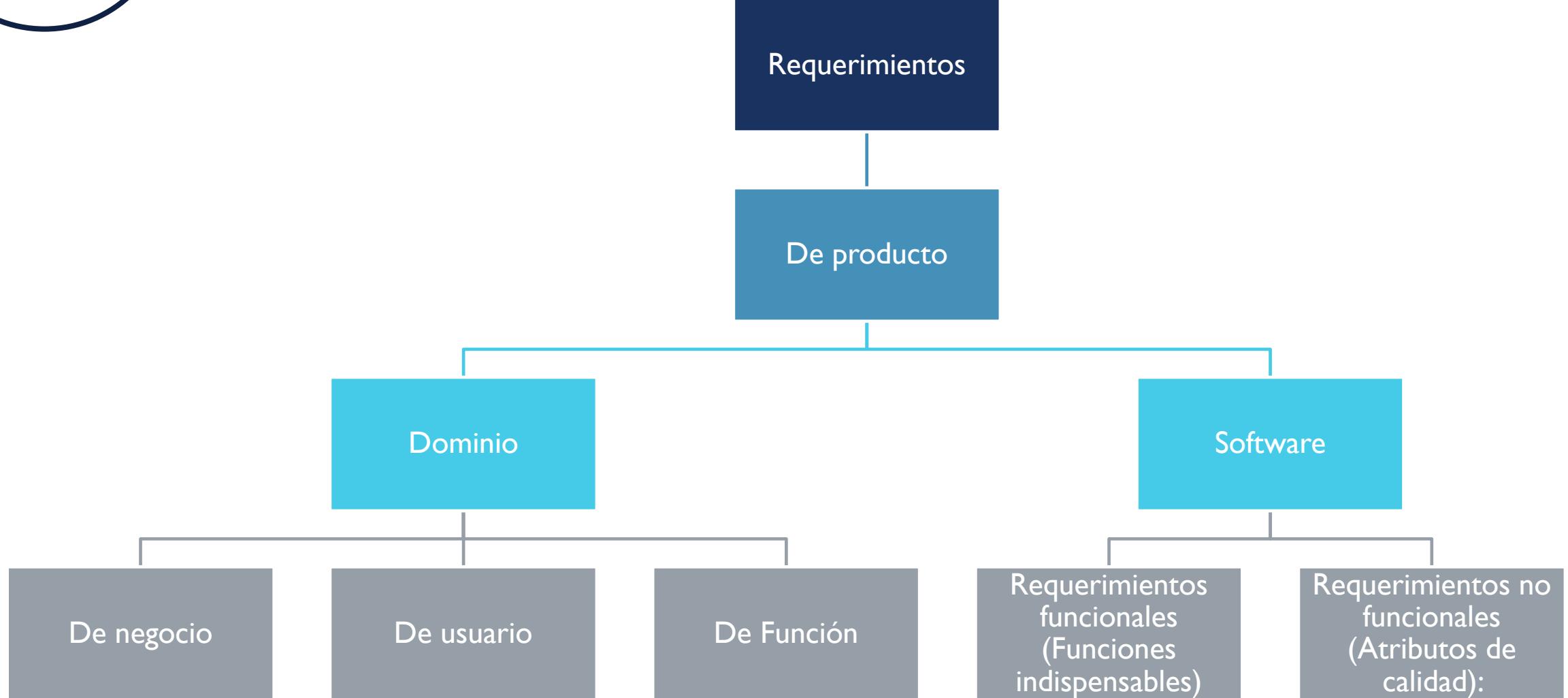
REQUERIMIENTOS: CLASIFICACIÓN

Andrés Armando Sánchez Martín

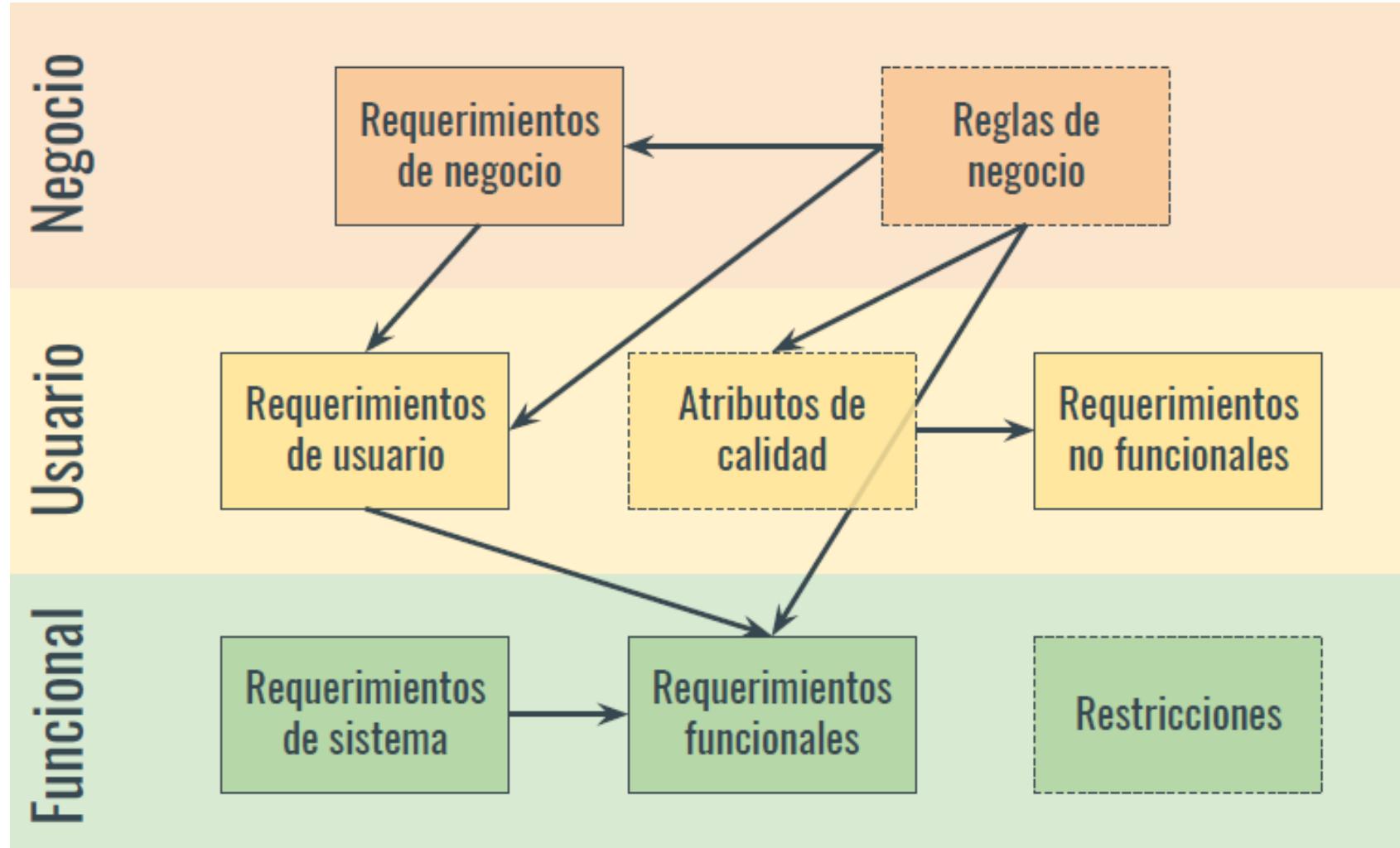


REQUERIMIENTOS: CLASIFICACIÓN

Andrés Armando Sánchez Martín



REQUERIMIENTOS: CLASIFICACIÓN





REQUERIMIENTOS: CLASIFICACIÓN

Requerimientos funcionales

“Como personal de enfermería quiero **ver la información del estado del paciente** para poder reaccionar a cualquier anomalía.”

Requerimientos no funcionales

“Como personal de enfermería quiero ver la información **en tiempo real** del estado del paciente para poder reaccionar a cualquier anomalía.”

RIESGOS

En la toma de requerimientos

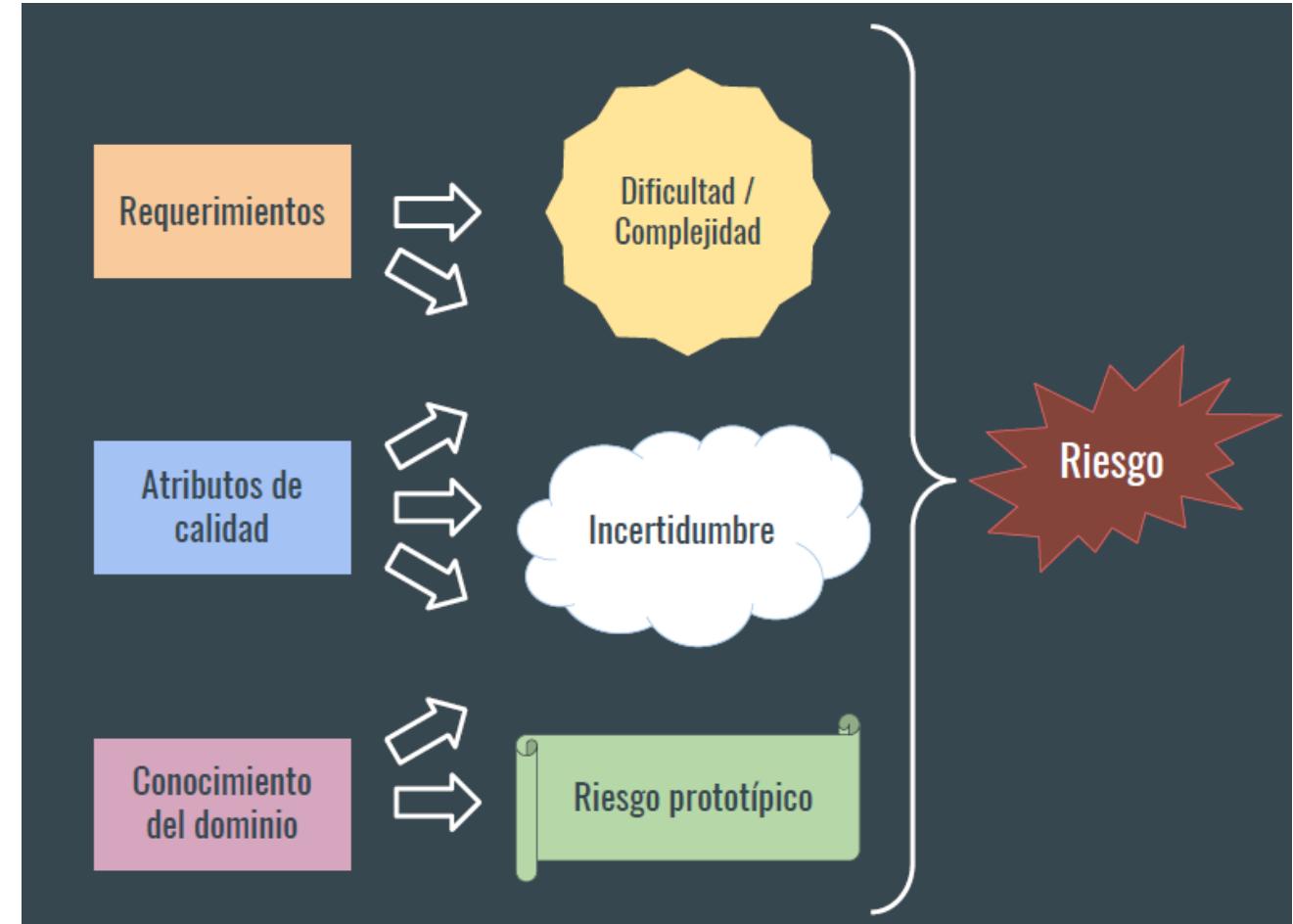
- dificultad / complejidad

En los atributos de calidad

- incertidumbre, cuanto mas incertidumbre hay, mas alto es el riesgo.

Conocimiento del dominio

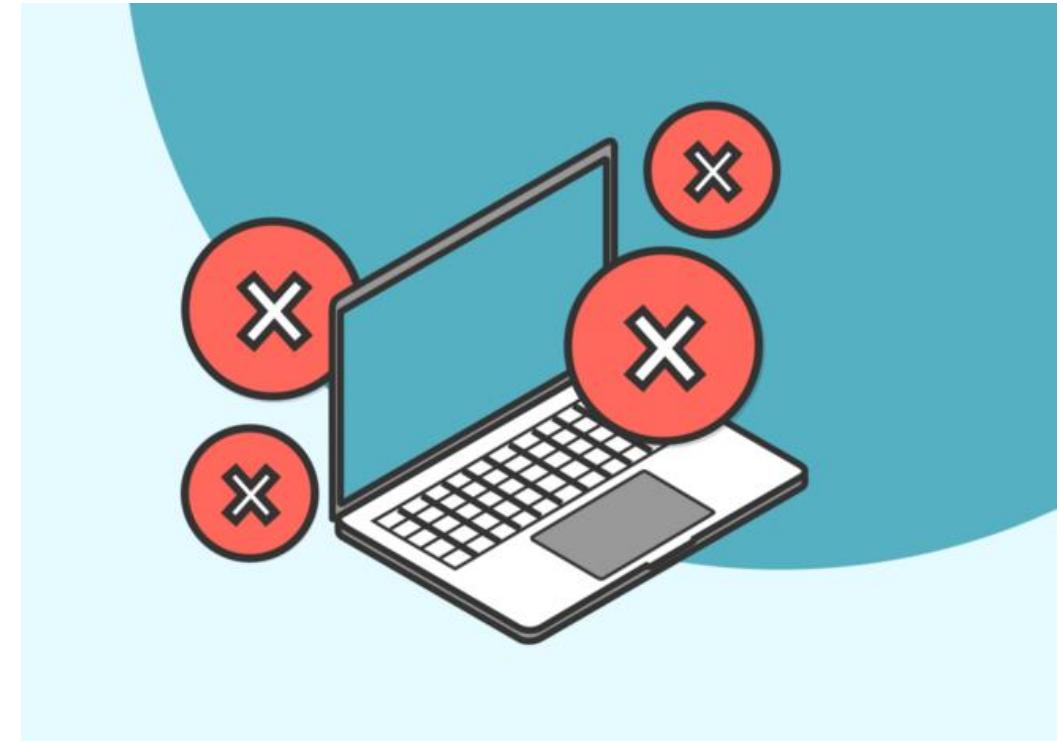
- Riesgo prototípico, son aquellos que podemos atacar de forma estándar.



RESTRICCIONES

Restricciones del sistema que vienen impuestas:

- Muy poco software tiene libertad total
- Pueden ser técnicas u organizativas
- Pueden surgir del cliente o también de la organización de desarrollo
- Limitan las alternativas a considerar para decisiones de diseño particulares
- Ejemplos:
 - Marcos de aplicaciones (frameworks)
 - Lenguajes de programación, ...
- Normalmente son tus “amigos”



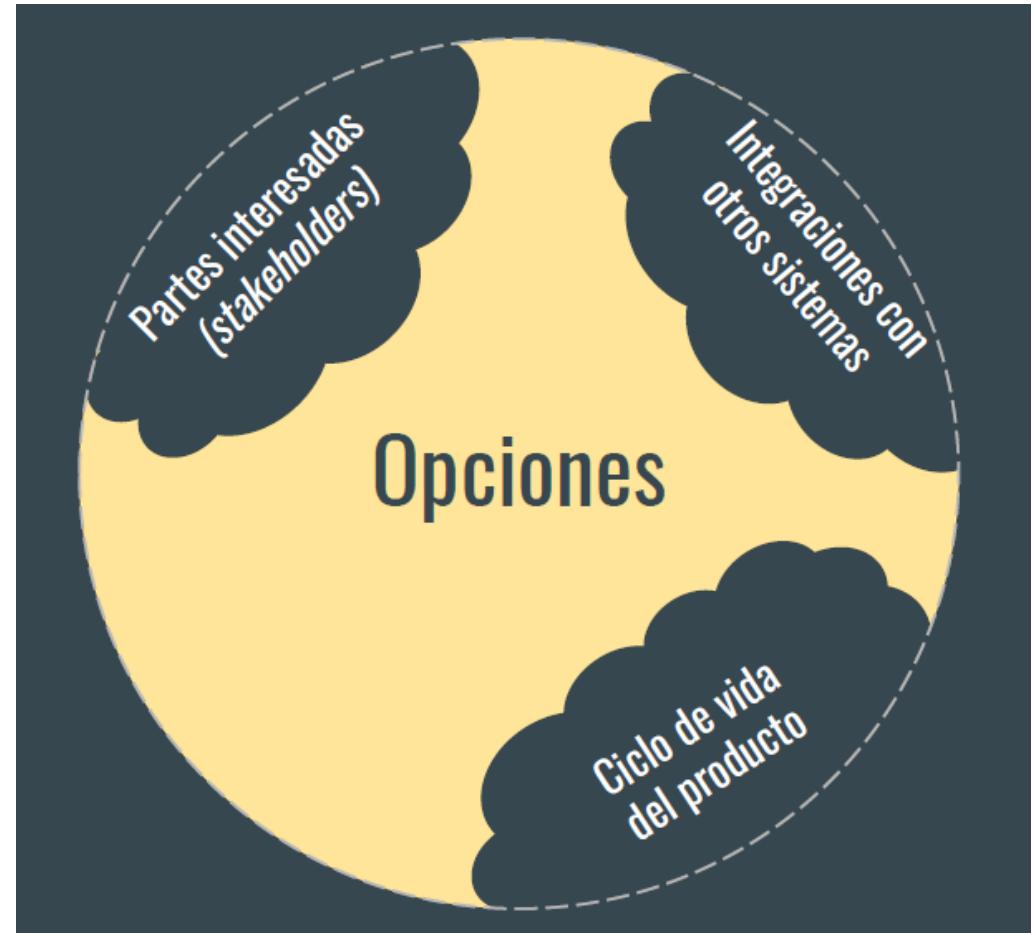
RESTRICCIONES

Las restricciones en el contexto de un proceso de desarrollo de software se refiere a las restricciones que limitan las opciones de diseño o implementaciones disponibles al desarrollar.

Los StakeHolders, nos pueden poner limitaciones relacionadas con su contexto de negocio, ejemplo:

- Las limitaciones legales, la implementación de un producto podría tener restricciones en algún país, y esto sería una limitante a considerar para el desarrollo del producto.
- Limitaciones técnicas, relacionadas con integraciones con otros sistemas.
- El ciclo de vida del producto, agregará limitaciones al producto, por ejemplo a medida que avanza el proceso de implementación el modelo de datos va a ser más difícil de modificar.

Nota: El arquitecto debe balancear entre los requerimiento y las restricciones



ATRIBUTOS DE CALIDAD

- Son características medibles de interés para usuarios o desarrolladores
- También conocidos como requisitos no-funcionales
 - Rendimiento, disponibilidad, modificabilidad, testabilidad, ...
 - También conocidos como -idades (-ities en inglés)
 - Pueden especificarse mediante escenarios
 - Técnica estímulo-respuesta: "Si ocurre un fallo interno durante la operación normal, el sistema reanuda la operación en menos de 30 segundos y no se pierden datos" Priorizados por: El cliente de acuerdo al éxito del sistema El arquitecto de acuerdo al riesgo técnico



ATRIBUTOS DE CALIDAD

- Los atributos de calidad determinan la mayoría de las decisiones de diseño en arquitectura
 - Si la única preocupación fuese la funcionalidad, un sistema monolíticos sería suficiente Sin embargo, es habitual ver:
 - Estructuras redundantes para fiabilidad
 - Estructuras concurrentes para rendimiento
 - Capas para modificabilidad ...
- Priorizados por:
 - El cliente de acuerdo al éxito del sistema
 - El arquitecto de acuerdo al riesgo técnico



Calidad

ATRIBUTOS DE CALIDAD

“Los atributos de calidad son las expectativas de usuario, en general implícitas, de cuán bien funcionará un producto.” - Software Requirements: 3rd Edition (Wiegert, Betty, 2013)

- Performance / Rendimiento
- Escalabilidad
- Modificabilidad
- Interoperabilidad
- Seguridad
- Disponibilidad
- Integración

Idoneidad Funcional

Eficiencia de ejecución

Compatibilidad

Usabilidad

Confiabilidad

Seguridad

Mantenibilidad

Portabilidad

ISO/IEC
25000:2014

MODELOS DE CALIDAD DE SOFTWARE

"los modelos de calidad son aquellos documentos que integran la mayor parte de las mejores prácticas, proponen temas de administración en los que cada organización debe hacer énfasis, integran diferentes prácticas dirigidas a los procesos clave y permiten medir los avances en calidad."

Estudio comparativo de los modelos y estándares de calidad del software (Scalone, 2006)

A nivel de proceso

- ITIL (1989)
- ISO/IEC 15504 (1993)
- Dromey (1995)
- PSP (1995)
- Bootstrap (1996)
- TSP (1996)
- IEEE 12207 (1996)
- COBIT 4,0 (1996)
- ISO 90003 (1997)
- CMMI (2000)
- ISO/IEC 20000 (2005)

A nivel de producto

- McCall (1977)
- GQM (1984)
- Boehm (1986)
- FURPS+ (1987)
- Gilb (1988)
- ISO 9126-1 (1991)
- SQAE (1997)
- WebQEM (1998)
- ISO 25000 (2005->2014)

MODELOS DE CALIDAD DE SOFTWARE

FURPS+

- *Functionality (Funcionalidad).*
- *Usability (Usabilidad).*
- *Reliability (Confiabilidad).*
- *Performance (Prestación).*
- *Supportability (Soporte).*
- + (Plus)

ISO/IEC 25000:2014

- Adecuación Funcional
- Eficiencia de Desempeño
- Compatibilidad
- Usabilidad
- Fiabilidad
- Seguridad
- Mantenibilidad
- Portabilidad

MODELOS DE CALIDAD DE SOFTWARE: FURPS+

Funcionalidad:

- Características
- Capacidades del sistemas
- Generalidades de las funciones
- Seguridad del sistema

Usabilidad

- Factores humanos.
- Factores estéticos.
- Consistencia de la interfaz.
- Documentación.
- Ayuda

Confiabilidad

- Exactitud de las salidas
- Tiempo medio de fallos
- Frecuencia de fallas
- Recuperabilidad frente a fallos.
- Grado de prevención.
- Capacidad de predicción



Rendimiento/Prestación

- Velocidad de procesamiento
- Eficiencia
- Consumo de recurso.
- Rendimiento efectivo total.
- Tiempo de respuesta.
- Disponibilidad

Capacidad de Soporte:

- Adaptabilidad.
- Extensibilidad.
- Mantenibilidad.
- Compatibilidad.
- Configurabilidad.
- Requisitos de instalación.
- Internacionalización

+

- Restricciones de Implementación
- Restricciones de Interfaz
- Empaquetamiento: para distribución
- Restricciones Legales
- Restricciones físicas



Pontificia Universidad
JAVERIANA
Bogotá

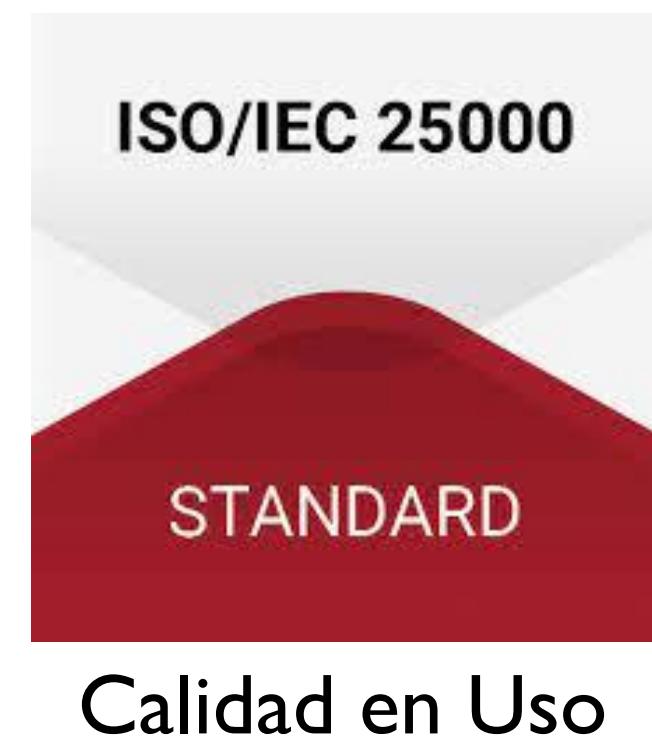
MODELOS DE CALIDAD DE SOFTWARE: ISO/IEC 25000:2014

Efectividad

Eficacia

Satisfacción

- Usefulness
- Trust
- Pleasure
- Comfort



Libre de Riesgos

- Economic risk mitigation
- Health and safety risk mitigation
- Environmental risk mitigation

Cobertura de Contexto

- Context completeness
- Flexibility

MODELOS DE CALIDAD DE SOFTWARE: ISO/IEC 25000:2014

Adecuación Funcional

- Completitud funcional.
- Corrección funcional.
- Pertinencia funcional.

Eficiencia de desempeño

- Comportamiento temporal.
- Utilización de recursos
- Capacidad

Compatibilidad

- Coexistencia
- Interoperabilidad

Usabilidad

- Capacidad para reconocer su adecuación
- Capacidad de aprendizaje
- Capacidad para ser usado
- Protección contra errores de usuario
- Estética de la interfaz de usuario
- Accesibilidad



ISO/IEC 25000

STANDARD

Calidad en Producto

Fiabilidad

- Madurez
- Disponibilidad
- Tolerancia a fallos.
- Capacidad de recuperación

Seguridad

- Confidencialidad
- Integridad
- No repudio
- Responsabilidad
- Autenticidad

Mantenibilidad

- Modularidad
- Reusabilidad
- Analizabilidad
- Capacidad para ser modificado
- Capacidad para ser probado

Portabilidad

- Adaptabilidad
- Capacidad para ser instalado
- Capacidad para ser reemplazado

IDONEIDAD FUNCIONAL

Comparación:

Requerimientos Funcionales y Funcionalidades implementadas

Completitud
Funcional

Exactitud
Funcional

Pertinencia
Funcional

IDONEIDAD FUNCIONAL

Comparación: resultado esperado y resultado obtenido

Completitud
Funcional

Exactitud
Funcional

Pertinencia
Funcional

IDONEIDAD FUNCIONAL

Comparación: objetivos cumplidos y objetivos esperados

Completitud
Funcional

Exactitud
Funcional

Pertinencia
Funcional

EFICIENCIA DE EJECUCIÓN

Comparación: Tiempo transcurrido entre pedido y respuesta y tiempo esperado o tiempo máximo tolerado

Tiempo a
Comportamiento

Uso de
Recursos

Capacidad

EFICIENCIA DE EJECUCIÓN

Comparación: Consumo de recursos y consumo esperado o tolerado de recursos

Tiempo a
Comportamiento

Uso de
Recursos

Capacidad

EFICIENCIA DE EJECUCIÓN

Comparación: Límite de tolerancia detectado y
límite de tolerancia esperado

Tiempo a
Comportamiento

Uso de
Recursos

Capacidad

COMPATIBILIDAD

Implementación de estándares y disponibilidad de esquemas: HATEOAS,
JSON Schema, SOAP, Open API

Interoperabilidad

Coexistencia

COMPATIBILIDAD

Cantidad de fallos por razones externas en un tiempo dado

Interoperabilidad

Coexistencia

USABILIDAD

Andrés Armando Sánchez Martín

Reconocimiento
de idoneidad

Curva de
aprendizaje

Operabilidad

Relación entre conceptos de dominio y acciones de sistema

USABILIDAD

Reconocimiento
de idoneidad

Curva de
aprendizaje

Operabilidad

Cantidad y tipo de consultas a soporte

USABILIDAD

Reconocimiento
de idoneidad

Curva de
aprendizaje

Operabilidad

Cantidad de pasos hasta lograr objetivos.
Métricas de conversión.

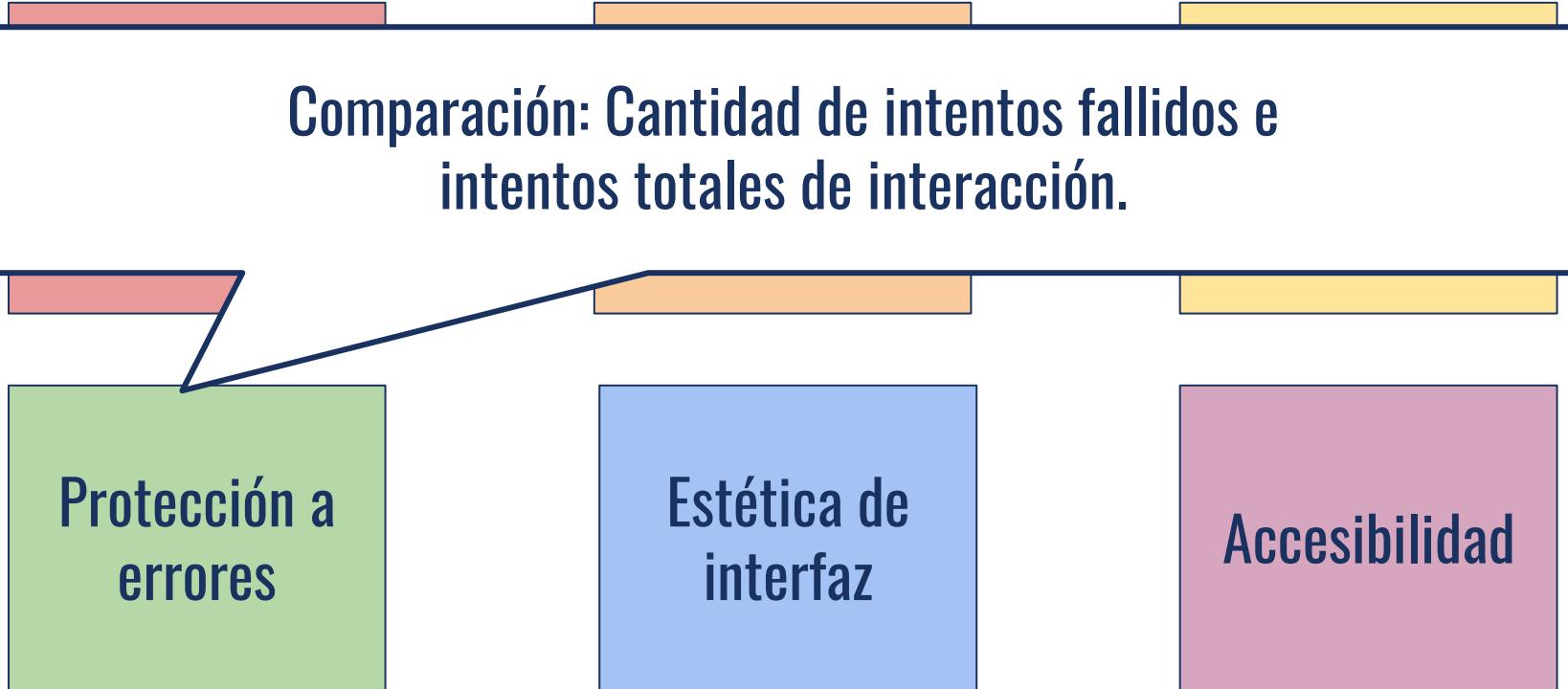
USABILIDAD

Comparación: Cantidad de intentos fallidos e intentos totales de interacción.

Protección a errores

Estética de interfaz

Accesibilidad



USABILIDAD

Encuestas de apreciación de estética

Protección a errores

Estética de interfaz

Accesibilidad

USABILIDAD

Andrés Armando Sánchez Martín

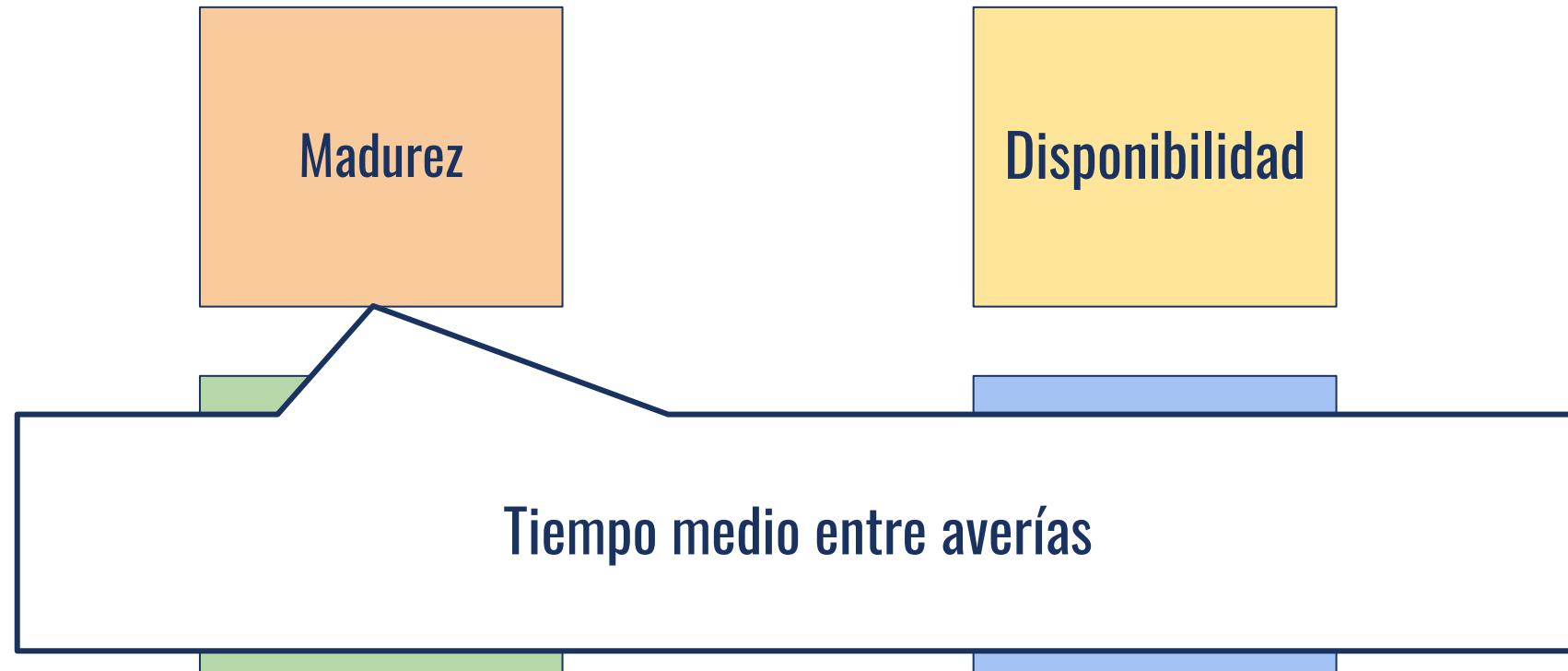
Protección a errores

Estética de interfaz

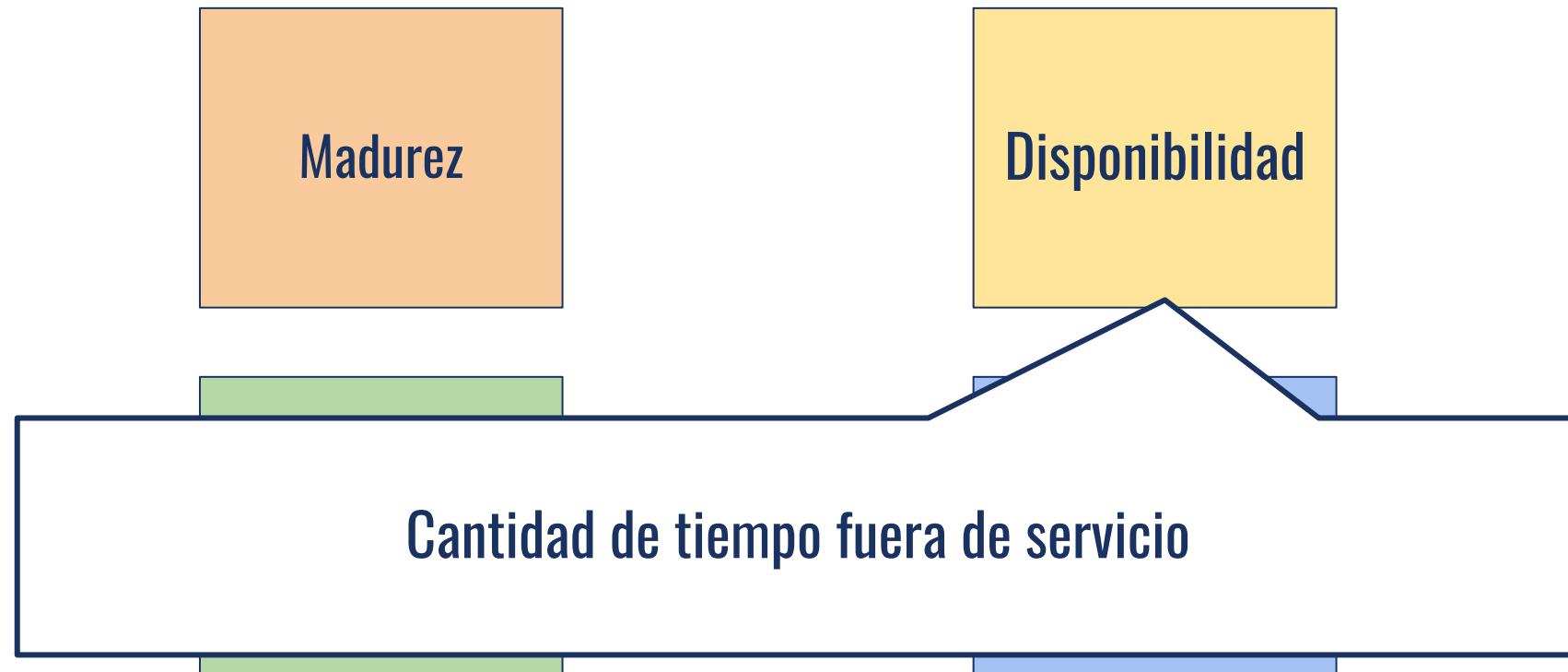
Accesibilidad

Adhesión a estándares

CONFIABILIDAD



CONFIABILIDAD



CONFIABILIDAD

Chaos testing

Tolerancia a
fallos

Capacidad de
recuperación

CONFIABILIDAD

Andrés Armando Sánchez Martín

Tiempo medio hasta la recuperación

Tolerancia a
fallos

Capacidad de
recuperación

SEGURIDAD

Penetration testing

Confidencialidad

Comprobación
de Hechos

Integridad

Traza de
responsabilidad

Autenticidad

MANTENIBILIDAD

Cobertura de código en tests y análisis estático de código

Modularidad

Capacidad de
Análisis

Reusabilidad

Capacidad de
Modificación

Capacidad de
Prueba

PORTABILIDAD

Cantidad de dependencias a entornos específicos.

Adaptabilidad

Capacidad de
Instalación

Capacidad de
Reemplazo

PORTABILIDAD

Requerimientos del entorno de despliegue

Adaptabilidad

Capacidad de
Instalación

Capacidad de
Reemplazo

PORTABILIDAD

Estándares o herramientas de carga o conversión de datos.
Implementación de interfaces comunes.

Adaptabilidad

Capacidad de
Instalación

Capacidad de
Reemplazo

BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué son los escenarios de calidad?
- ¿Cómo se miden los atributos de calidad?
- ¿Qué es ADD (Diseño Centrado en Atributos de Calidad)?





(1306)

ARQUITECTURA DE SOFTWARE

3.1 Atributos de Calidad 2

Agenda

- Atributos de Calidad
- Escenarios de Calidad
- Métricas de Calidad
- Diseño Centrado en Atributos de Calidad



ATRIBUTOS DE CALIDAD

“Los atributos de calidad son las expectativas de usuario, en general implícitas, de cuán bien funcionará un producto.” - Software Requirements: 3rd Edition (Wiegers, Betty, 2013)

- Performance / Rendimiento
- Escalabilidad
- Modificabilidad
- Interoperabilidad
- Seguridad
- Disponibilidad
- Integración

Idoneidad Funcional

Eficiencia de ejecución

Compatibilidad

Usabilidad

Confiabilidad

Seguridad

Mantenibilidad

Portabilidad

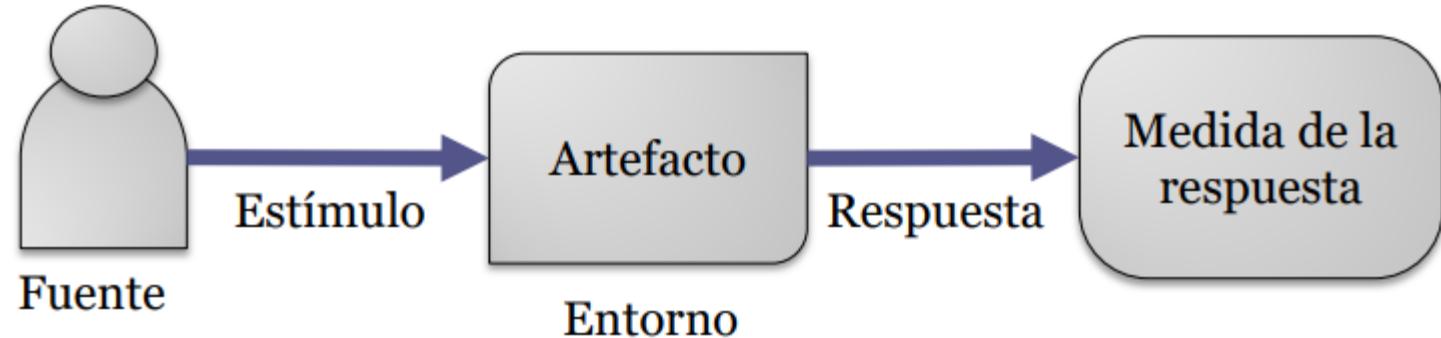
ISO/IEC
25000:2014

ATRIBUTOS DE CALIDAD

- 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
- La calidad es el 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

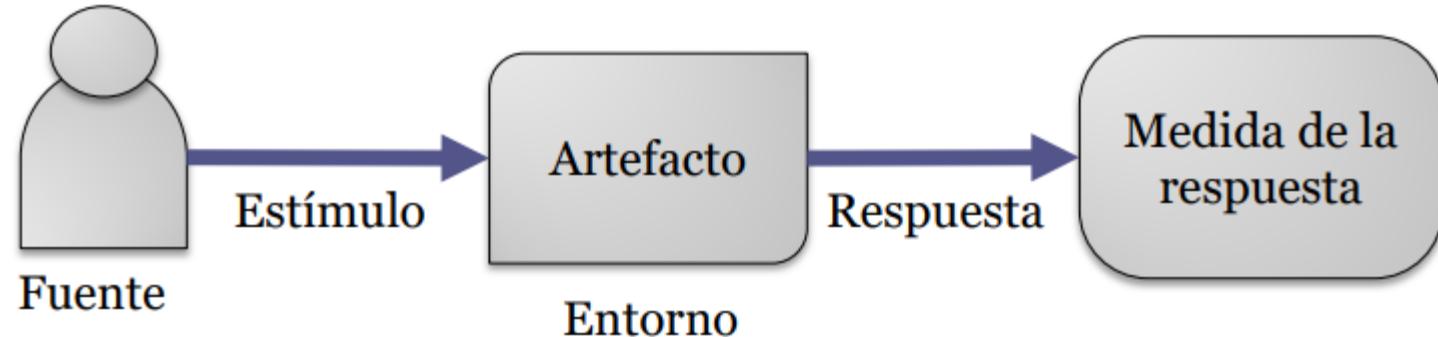
ESCENARIO DE CALIDAD

- 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



ESCENARIO DE CALIDAD: COMPONENTES

- 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:TIPOS

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)

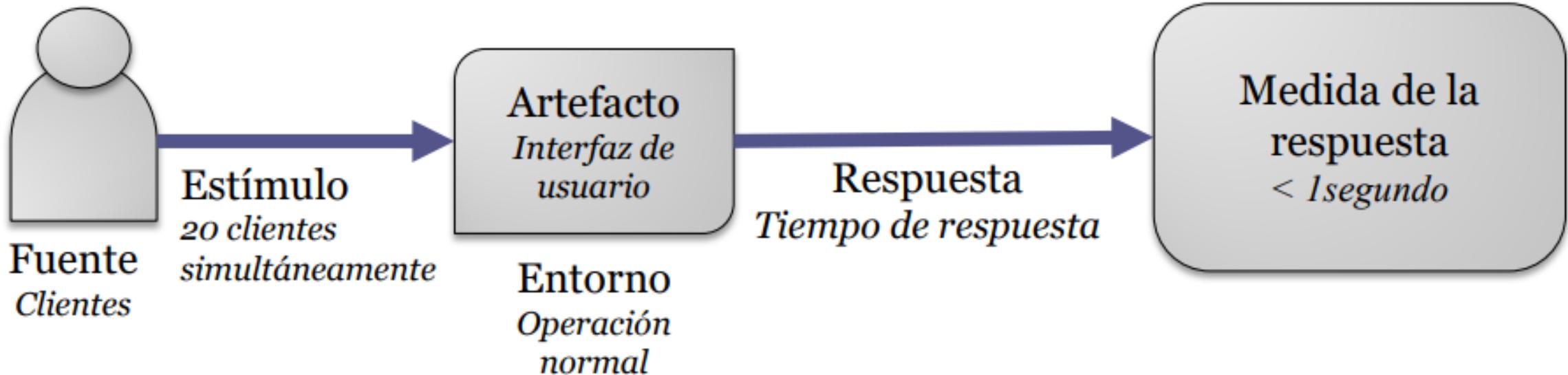
- Cambio en cualquier componente dentro del Sistema, en entorno o la infraestructura operacional

Escenarios de fallo:

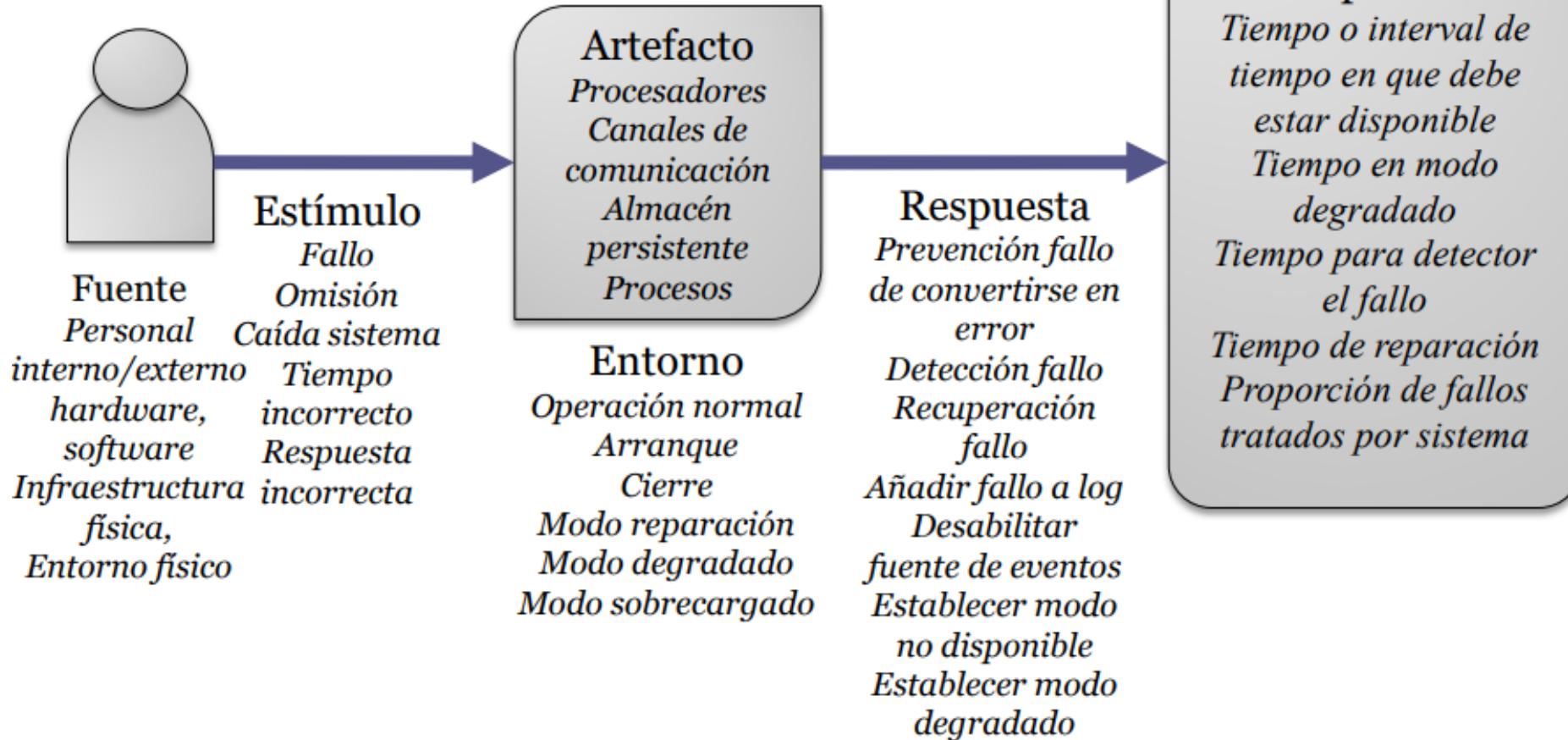
- Alguna parte del Sistema, infraestructura o entorno falla

ESCENARIO DE CALIDAD: EJEMPLO

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



ESCENARIO DE CALIDAD: EJEMPLO DISPONIBILIDAD



ESCENARIO DE CALIDAD: PRIORIZACIÓN

- 2 valores (Low/Medium/High)
- Ejemplo:
- Cómo es de importante para el éxito (cliente)
- Cómo de difícil es de alcanzar (arquitecto)

Ref	AC	Escenario	Prioridad
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 lo 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

MÉTRICAS DE CALIDAD: ¿CÓMO SABER SI MEJORAMOS?

Si no se mide

No se controla

Si no se controla

No se mejora



MÉTRICAS DE CALIDAD

- Los escenarios de AC requieren medir respuestas
- Problemas habituales:
 - Muchos AC tienen significados vagos
 - Ejemplo: desplegabilidad, escalabilidad
- Definiciones muy diversas
 - No hay definiciones aceptadas universalmente
- Demasiado compuestos
 - Normalmente, un AC está compuesto de varias características

Según el contexto

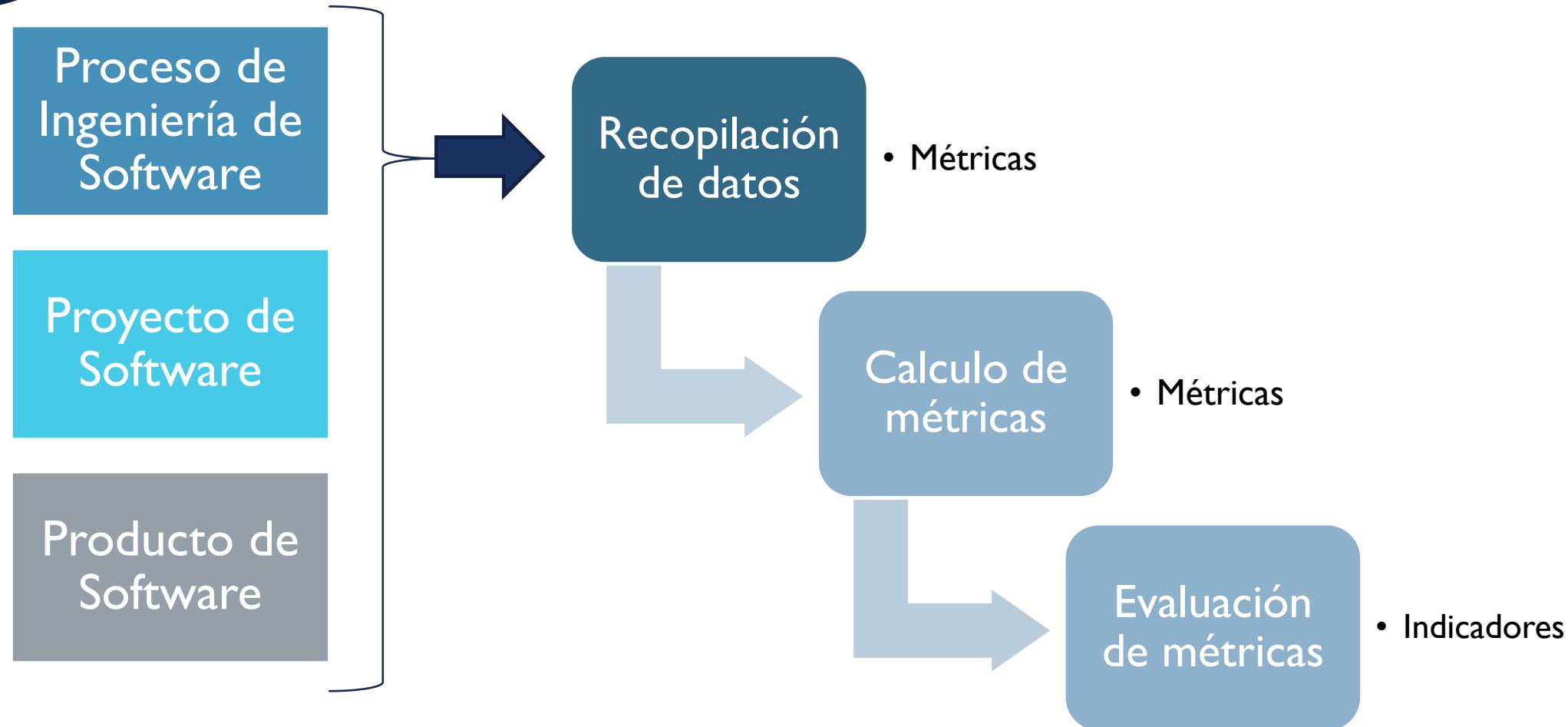
Métricas de Proceso

Métricas de Proyecto

Métricas de Producto

MÉTRICAS DE ATRIBUTOS DE CALIDAD: PROCESO

Andrés Armando Sánchez Martín



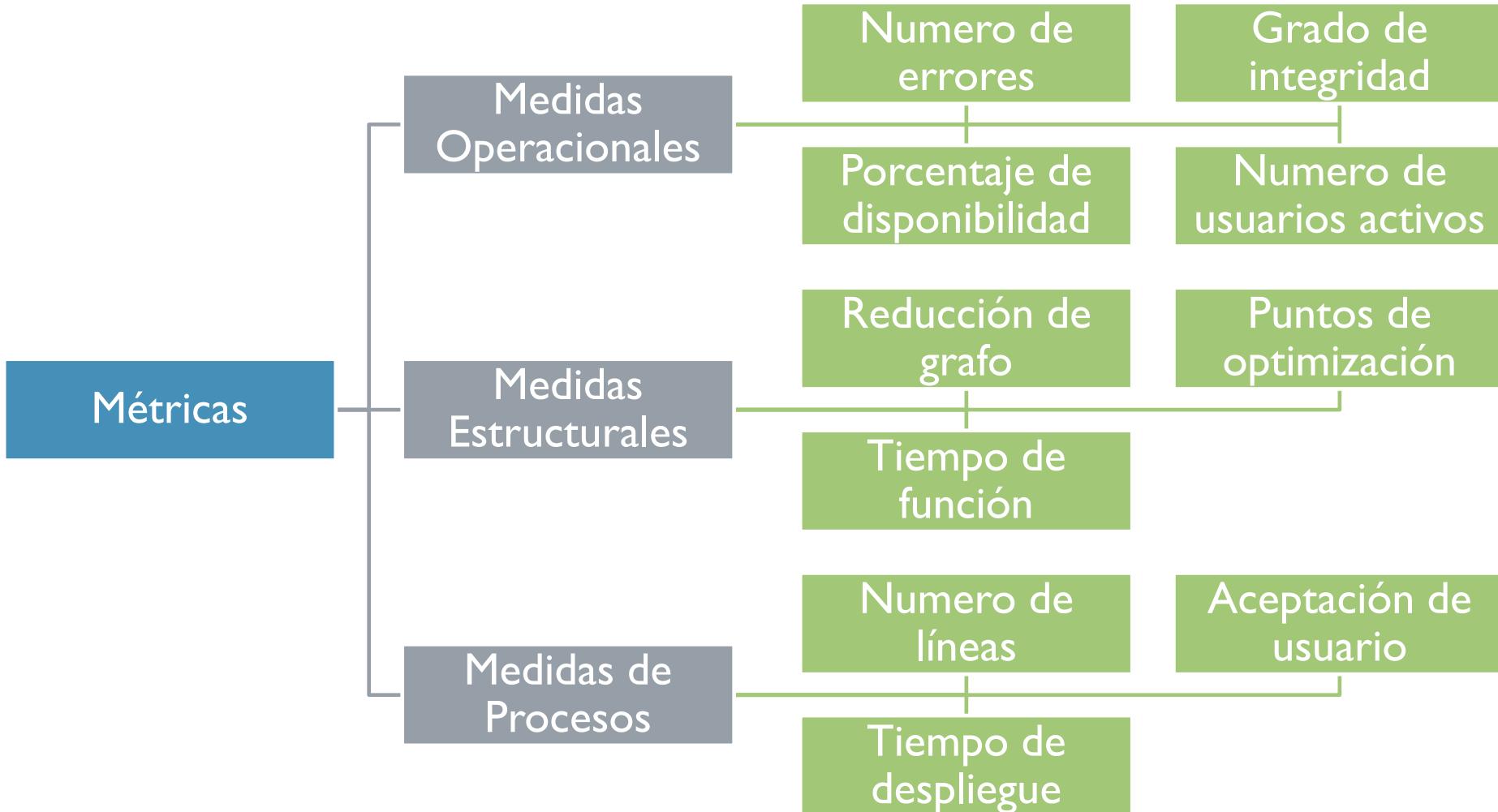
MÉTRICAS DE ATRIBUTOS DE CALIDAD

Intentar medir de forma objetiva aspectos de la calidad

Numerosas métricas para diferentes aspectos



MÉTRICAS DE ATRIBUTOS DE CALIDAD: EJEMPLOS

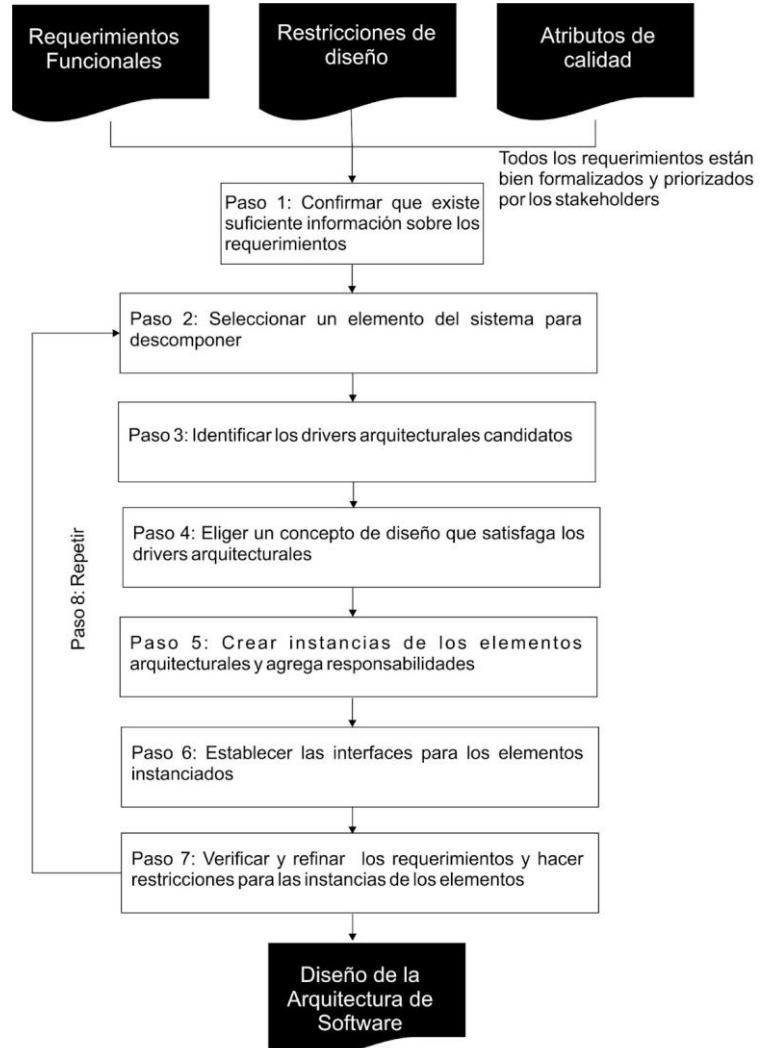


DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD)

El método ADD es un enfoque a la definición de una arquitectura de software en el que se basa el proceso de diseño de los requisitos del software de atributos de calidad. ADD es un proceso de diseño recursivo que descompone un sistema o elemento del sistema mediante la aplicación de tácticas arquitectónicas y los patrones que satisfacen los drivers.

ADD sigue esencialmente el ciclo “Plan, Do, and Check”:

- Plan: Los atributos de calidad y las restricciones de diseño son consideradas para seleccionar qué tipos de elementos se utilizan en la arquitectura.
- Do: Los elementos son instanciados para satisfacer los atributos de calidad, así como los requisitos funcionales.
- Check: El resultado es analizado para determinar si los requisitos son satisfechos.

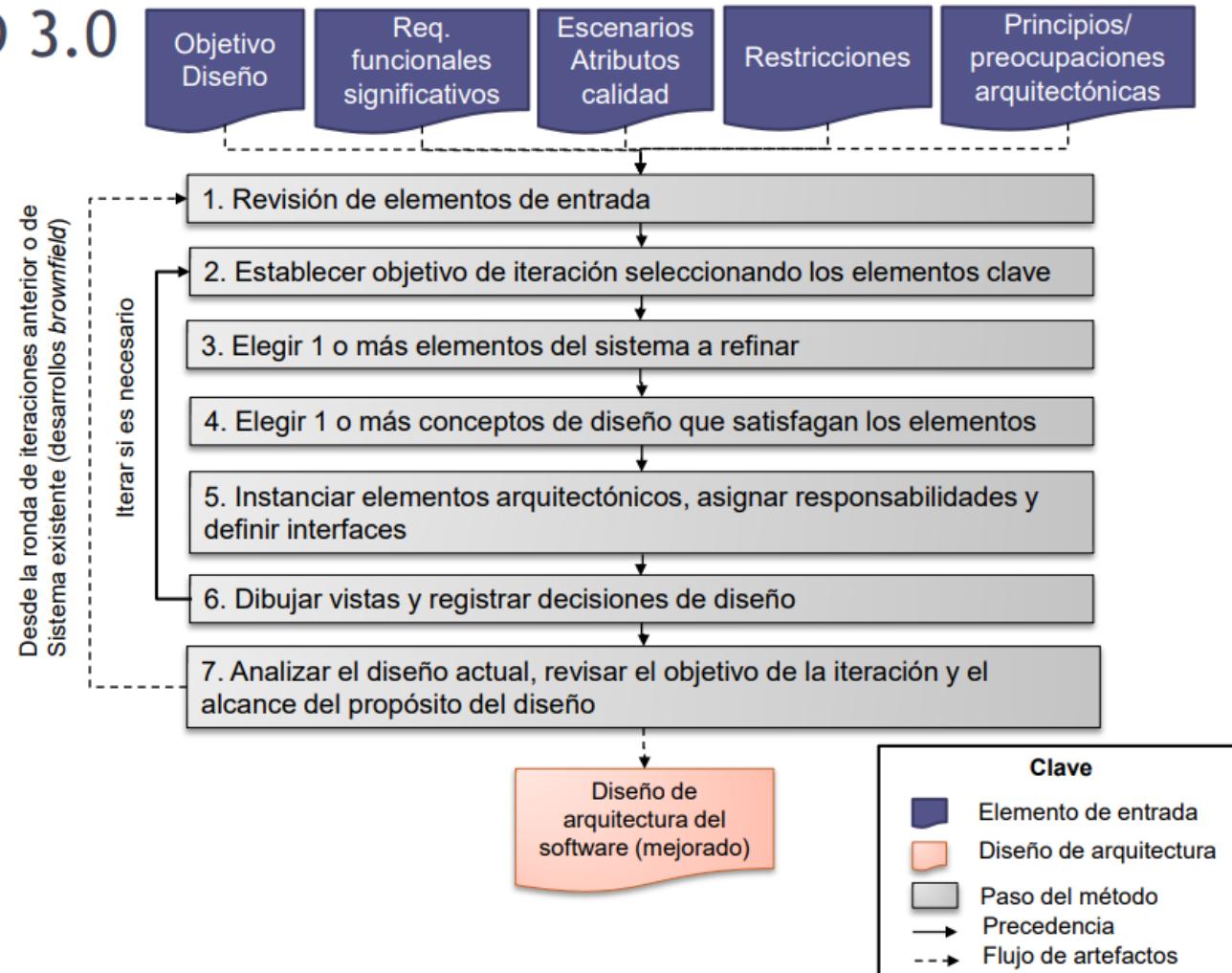


DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD)

- 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

DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD)

ADD 3.0



DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD): ESCENARIOS (FRAMEWORKS) Y TÁCTICAS

Framework de diseño orientado a atributos: Busca priorizar en el diseño, la solución e implementación los requerimientos, riesgos y atributos de calidad que fueron definidos para el proyecto. Plantea una estructura de escenarios y tácticas, en dónde cada escenario relaciona un atributos de calidad con distintas técnicas de implementación buscando mejorar la calidad.

1	Estímulo: cualquier hecho que afecte la calidad
2	Técnicas de respuesta: buscará controlar la respuesta al estímulo
3	Respuesta: respuesta esperada o caso de éxito al implementar la técnica



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition. 2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer. 2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin. Wiley. 2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley. 2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Cuál es la tensión entre los Atributos de Calidad?
- ¿Cómo se validan las decisiones en ADD?
- ¿Qué son los Acuerdos (Trade-offs)?





(1306)

ARQUITECTURA DE SOFTWARE

3.2 Atributos de Calidad 3

Agenda

- Atributos de Calidad
- Tensión entre Atributos
- Acuerdos
- Gestión de Atributos de Calidad
- Evolución de Arquitectura
- Caso de Estudio Atributos de Calidad





ATRIBUTOS DE CALIDAD

“Los atributos de calidad son las expectativas de usuario, en general implícitas, de cuán bien funcionará un producto.” - Software Requirements: 3rd Edition (Wiegers, Betty, 2013)

Performance / Rendimiento

Escalabilidad

Modificabilidad

Interoperabilidad

Seguridad

Disponibilidad

Integración

Idoneidad Funcional

Eficiencia de ejecución

Compatibilidad

Usabilidad

Confiabilidad

Seguridad

Mantenibilidad

Portabilidad

ISO/IEC
25000:2014

TENSIÓN ENTRE ATRIBUTOS

Todos los ACs son buenos ...pero el valor depende del Proyecto y Stakeholder

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

Los 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

2 consideraciones:

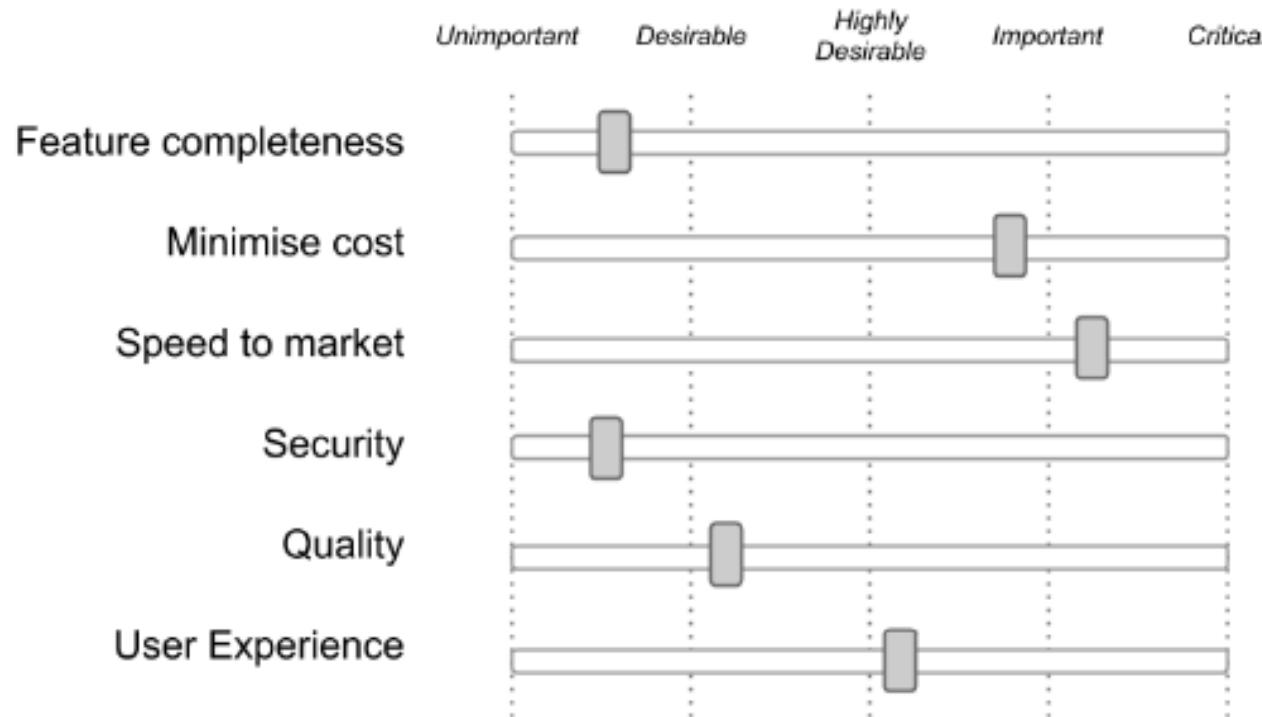
- Los ACs por sí mismos no son suficientes
 - No son operacionales
 - Ejemplo: No aporta mucho decir que un Sistema debe ser modifiable, o tener alta disponibilidad
- El vocabulario para describir ACs es muy variado
 - Es necesario describir cada atributo por separado

Escenarios Calidad: Forma común de especificar requisitos de AC



¡No hay sistema o arquitectura perfecta!

ACUERDOS (TRADE-OFFS)



- Decisiones significativas desde el punto de vista de la arquitectura del software
- Normalmente, decisiones que afectan a:
 - Atributos de calidad
 - Estructura
 - Dependencias
 - Interfaces
 - Técnicas de construcción

ACUERDOS (TRADE-OFFS): MATRIZ DE PUNTOS DE ACUERDO POR AC

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability						+		+				
Efficiency		-	-	-	-	-	-	-	-	-	-	-
Flexibility	-	-	-	+	+	+			+			
Integrity	-		-				-		-	-		
Interoperability	-	+	-		+							
Maintainability	+	-	+			+			+			
Portability		-	+	+	-		+		+	+	-	
Reliability	+	-	+		+			+	+	+		
Reusability		-	+	-	+	+	+	-		+		
Robustness	+	-				+					+	
Testability	+	-	+		+	+					+	
Usability		-						+	-			

Leyenda:

[-] Aumentar el atributo de la fila impacta negativamente a los atributos en las columnas

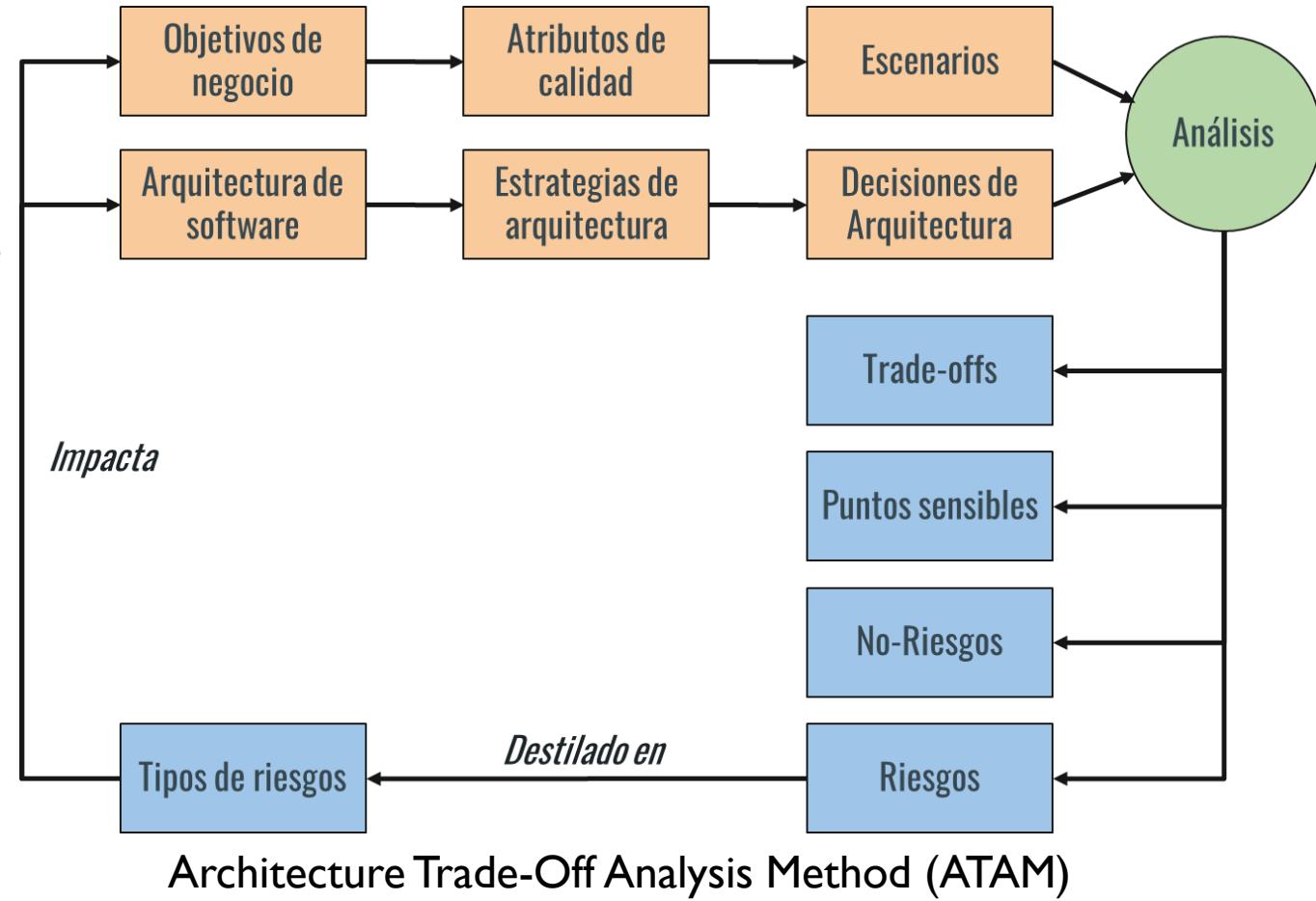
[+] Aumentar el atributo de la fila tiene un impacto positivo sobre los atributos en las columnas

[] No existe mucha interacción entre los atributos

GESTIÓN DE ATRIBUTOS DE CALIDAD

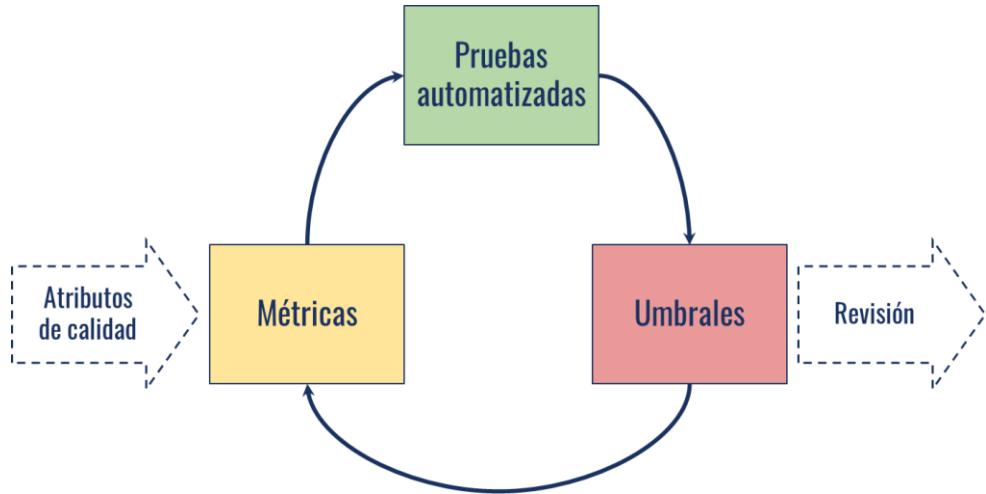
Es importante tener en cuenta que durante el diseño de un sistema no puede lograrse la satisfacción todos los atributos de calidad al 100%. Satisfacer cierto atributo de calidad puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean satisfacer. En el contexto de diseño de sistemas a este fenómeno se le conoce como “trade-off”. Un arquitecto de software, idealmente, debería conocer sobre “trade-offs”.

- En arquitecturas evolutivas:
 - Función de encaje (fitness): mecanismo que proporciona una valoración de la integridad objetiva alguna combinación de atributos de calidad
 - Medir valores de AC y evolucionar la arquitectura



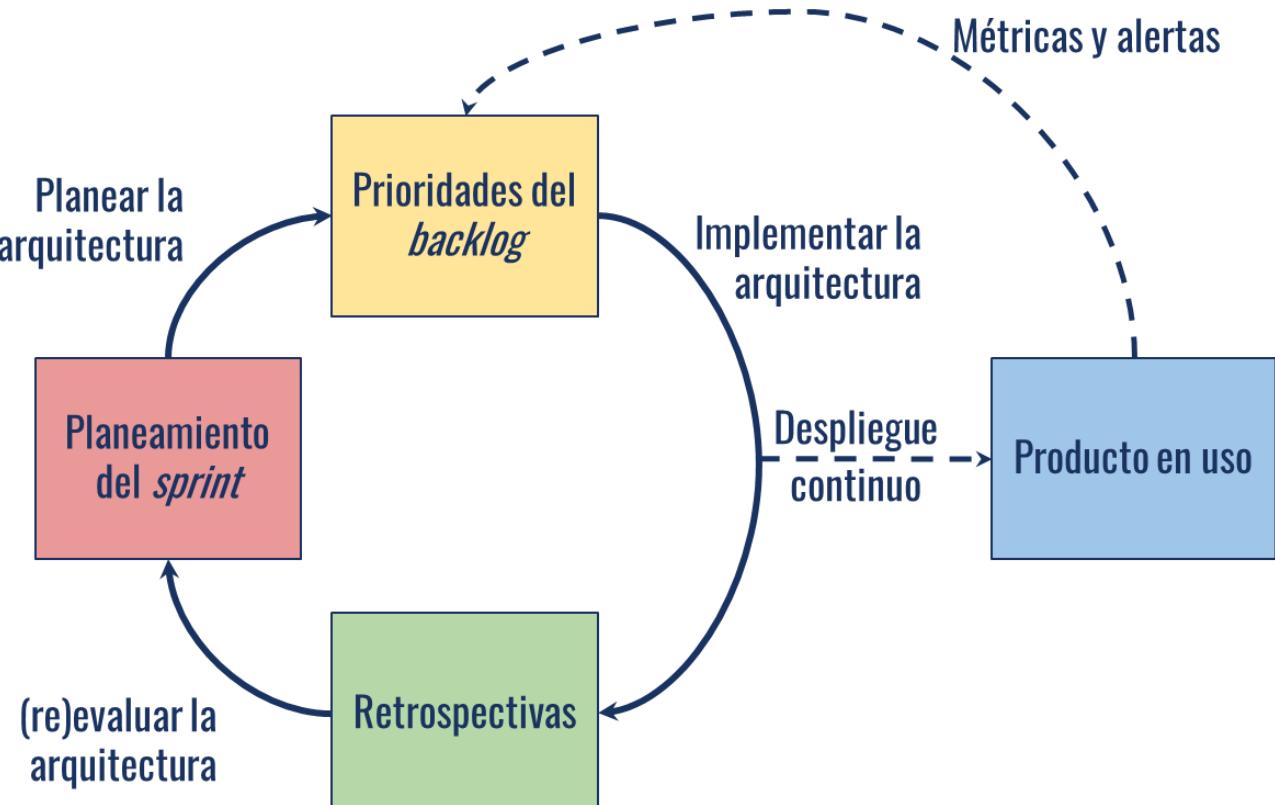
EVOLUCIÓN DE ARQUITECTURA

Andrés Armando Sánchez Martín



EVALUACIÓN DE MÉTRICAS

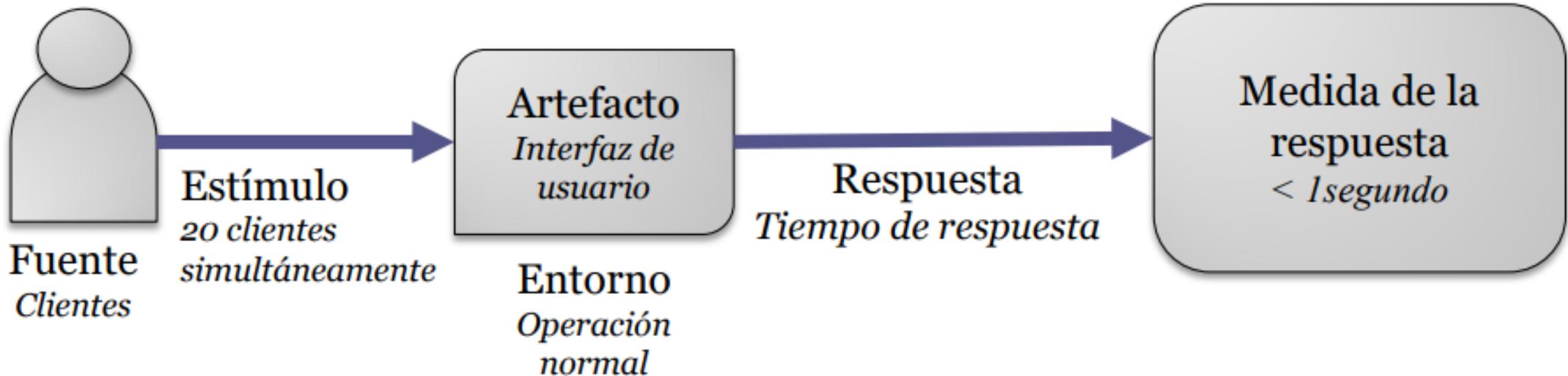
8



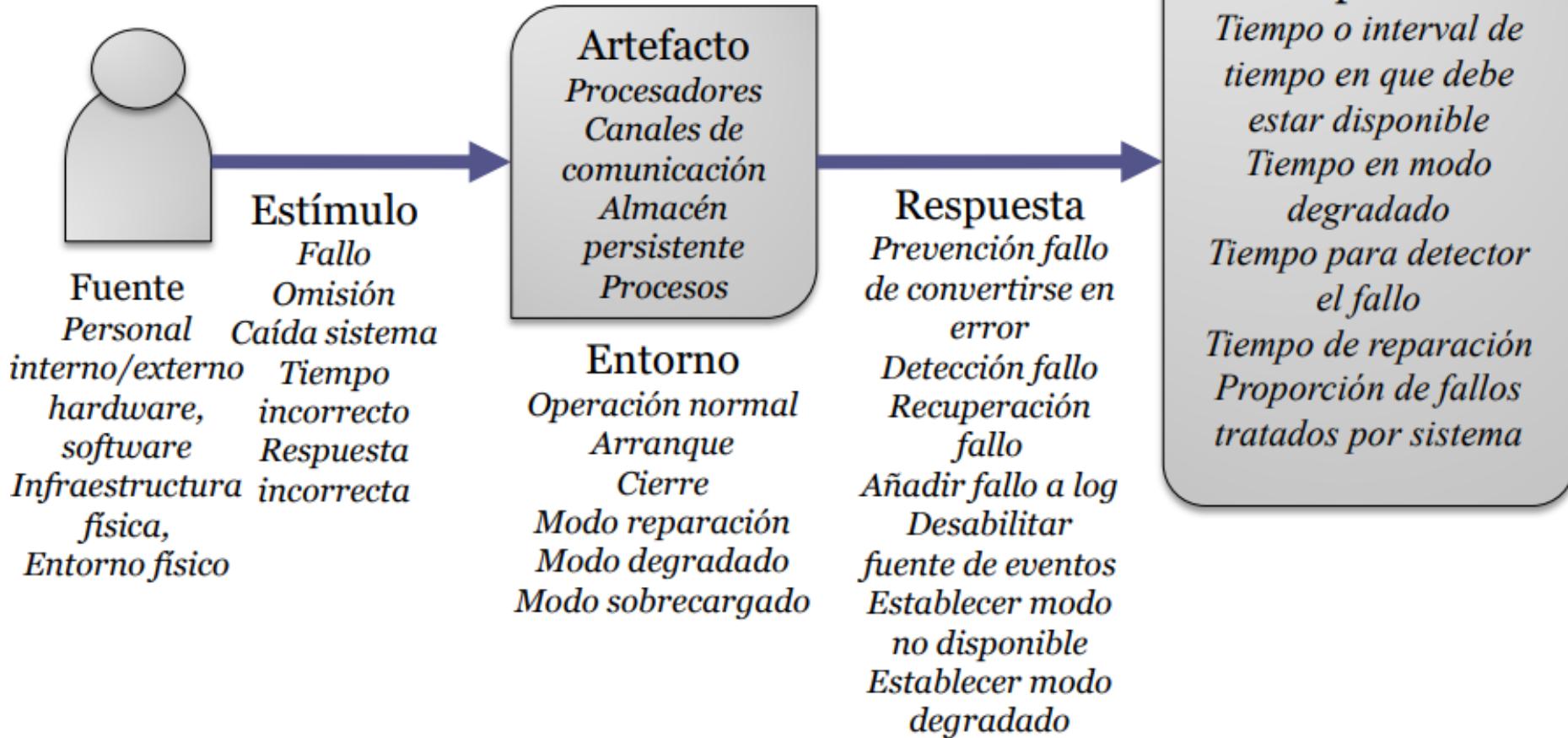
ARQUITECTURA EN EVOLUCIÓN

ESCENARIO DE CALIDAD: EJEMPLO

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



ESCENARIO DE CALIDAD: EJEMPLO DISPONIBILIDAD



DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD): ESCENARIOS (FRAMEWORKS) Y TÁCTICAS

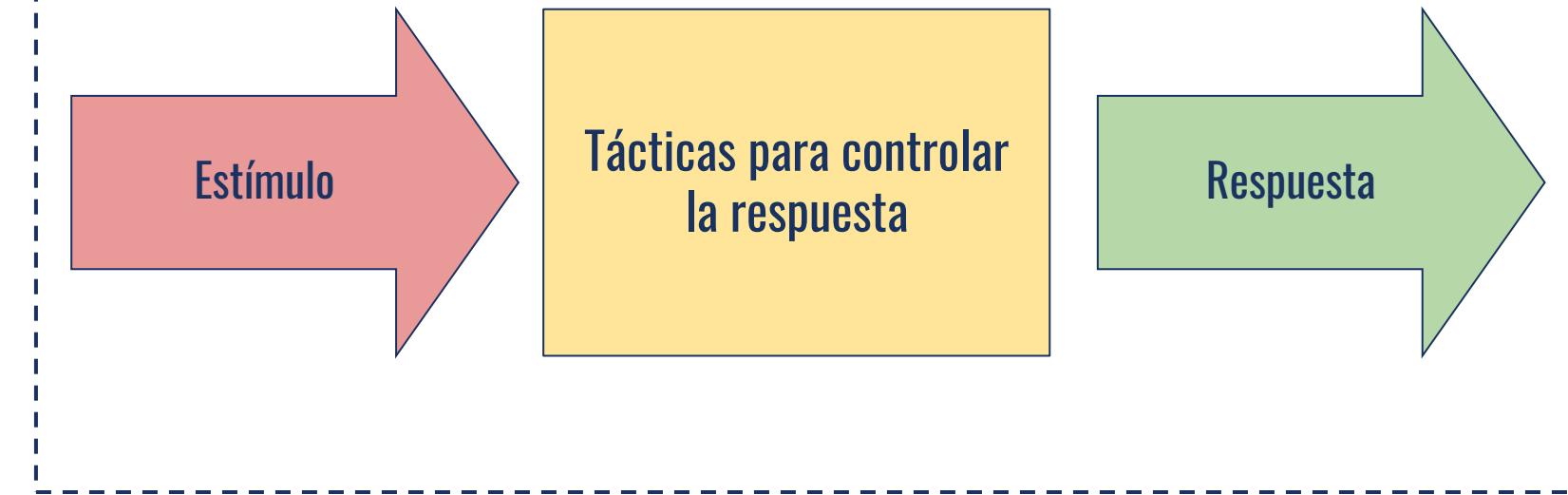
Framework de diseño orientado a atributos: Busca priorizar en el diseño, la solución e implementación los requerimientos, riesgos y atributos de calidad que fueron definidos para el proyecto. Plantea una estructura de escenarios y tácticas, en dónde cada escenario relaciona un atributos de calidad con distintas técnicas de implementación buscando mejorar la calidad.

1	Estímulo: cualquier hecho que afecte la calidad
2	Técnicas de respuesta: buscará controlar la respuesta al estímulo
3	Respuesta: respuesta esperada o caso de éxito al implementar la técnica

DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD): ESCENARIOS (FRAMEWORKS) Y TÁCTICAS

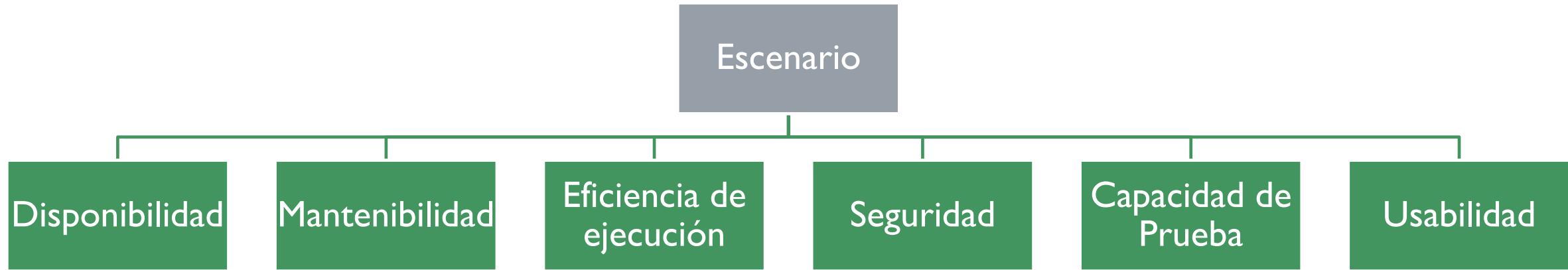
Andrés Armando Sánchez Martín

Escenario: Atributo de Calidad X



Algunos patrones y estilos ya implementan técnicas para el manejo de atributos

ESCENARIOS Y TÁCTICAS: CLASIFICACIÓN



Se clasifican según el atributo de calidad. Aunque no hay definición para todos los atributos de calidad definidos en la ISO/IEC 25000

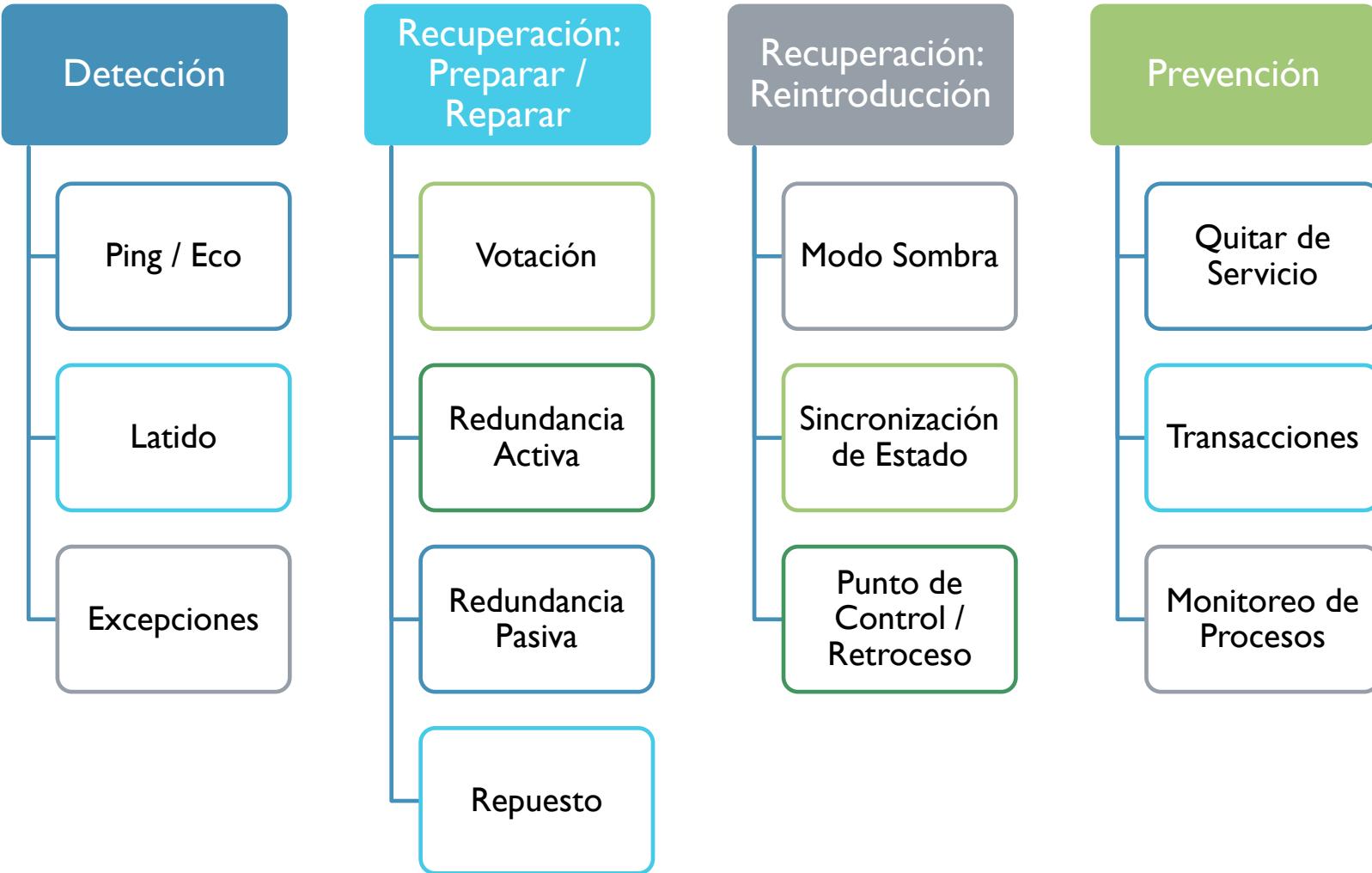
ESCENARIOS Y TÁCTICAS: DISPONIBILIDAD

Escenario: Disponibilidad



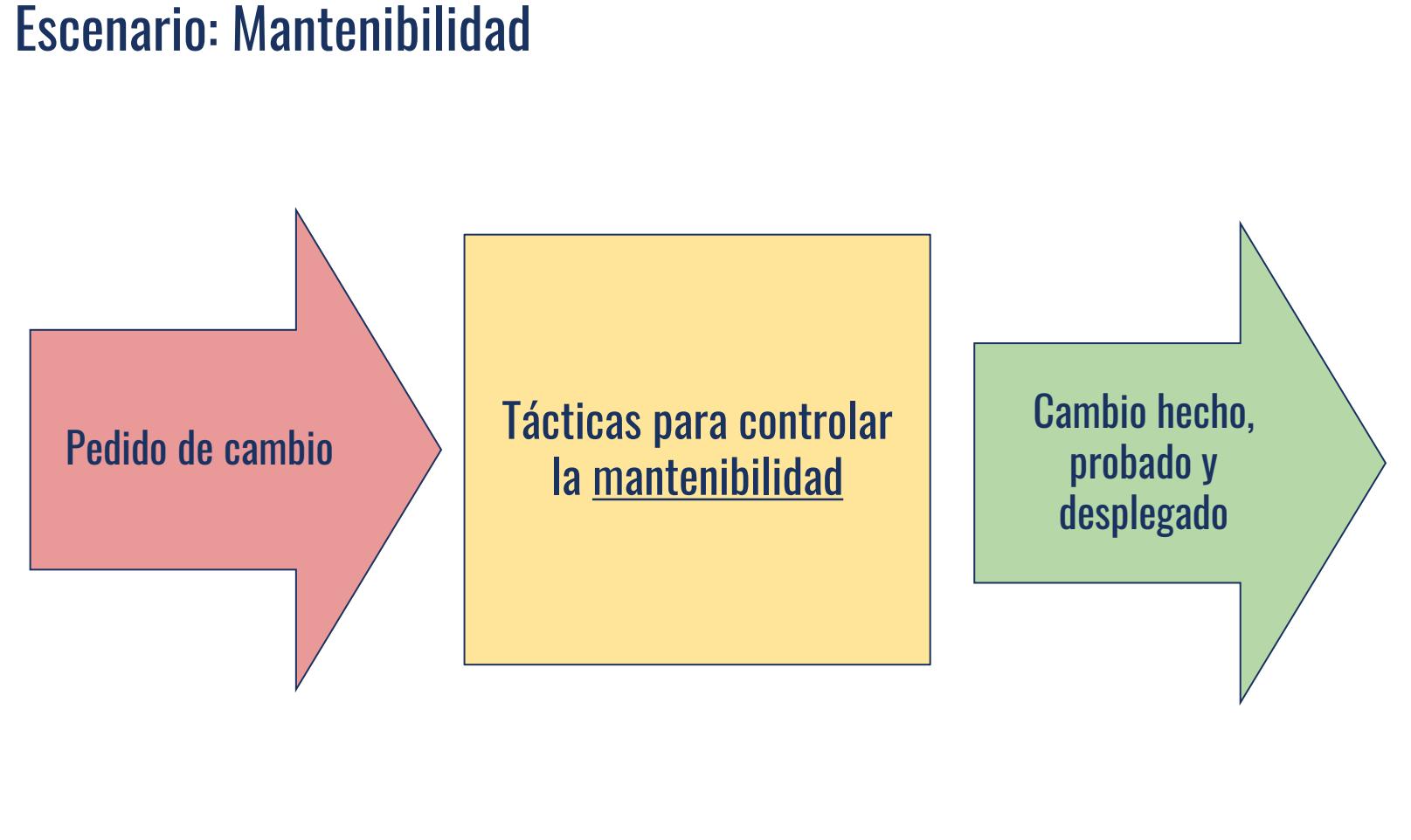
ESCENARIOS Y TÁCTICAS: DISPONIBILIDAD

Andrés Armando Sánchez Martín



ESCENARIOS Y TÁCTICAS: MANTENIBILIDAD

Escenario: Mantenibilidad



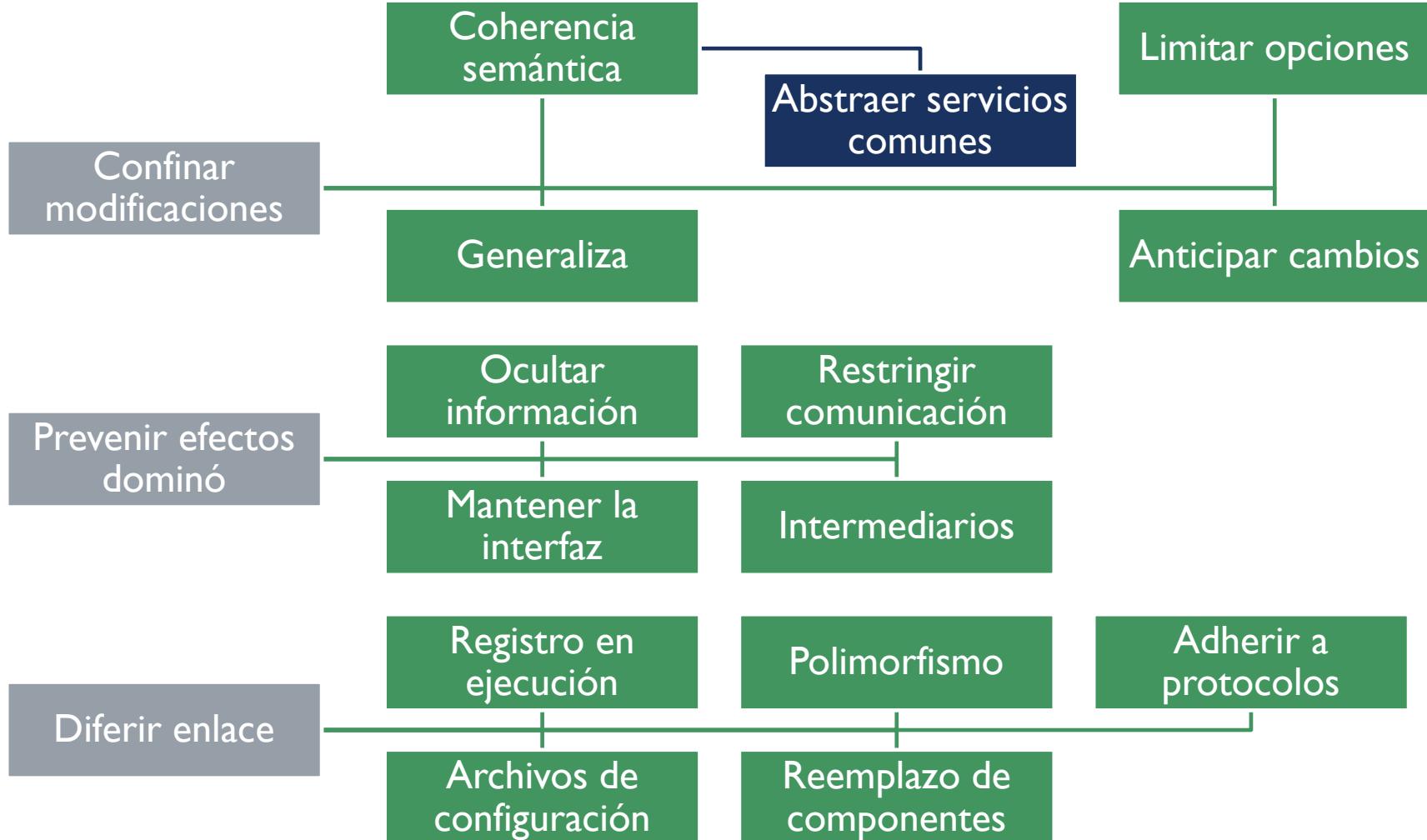
Pedido de cambio

Táticas para controlar
la mantenibilidad

Cambio hecho,
probado y
desplegado

ESCENARIOS Y TÁCTICAS: MANTENIBILIDAD

Andrés Armando Sánchez Martín



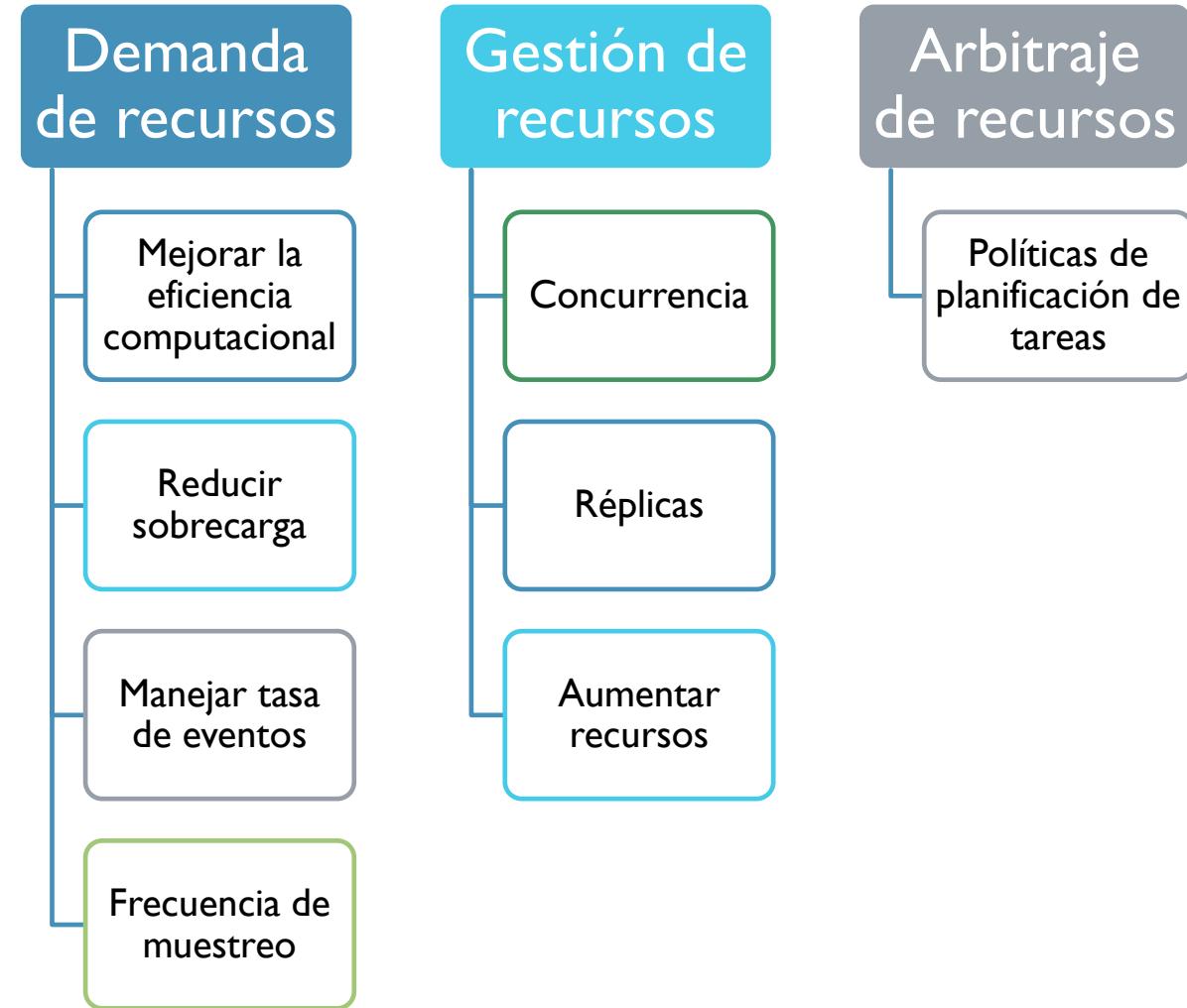
ESCENARIOS Y TÁCTICAS: EFICIENCIA DE EJECUCIÓN

Andrés Armando Sánchez Martín

Escenario: Eficiencia de ejecución

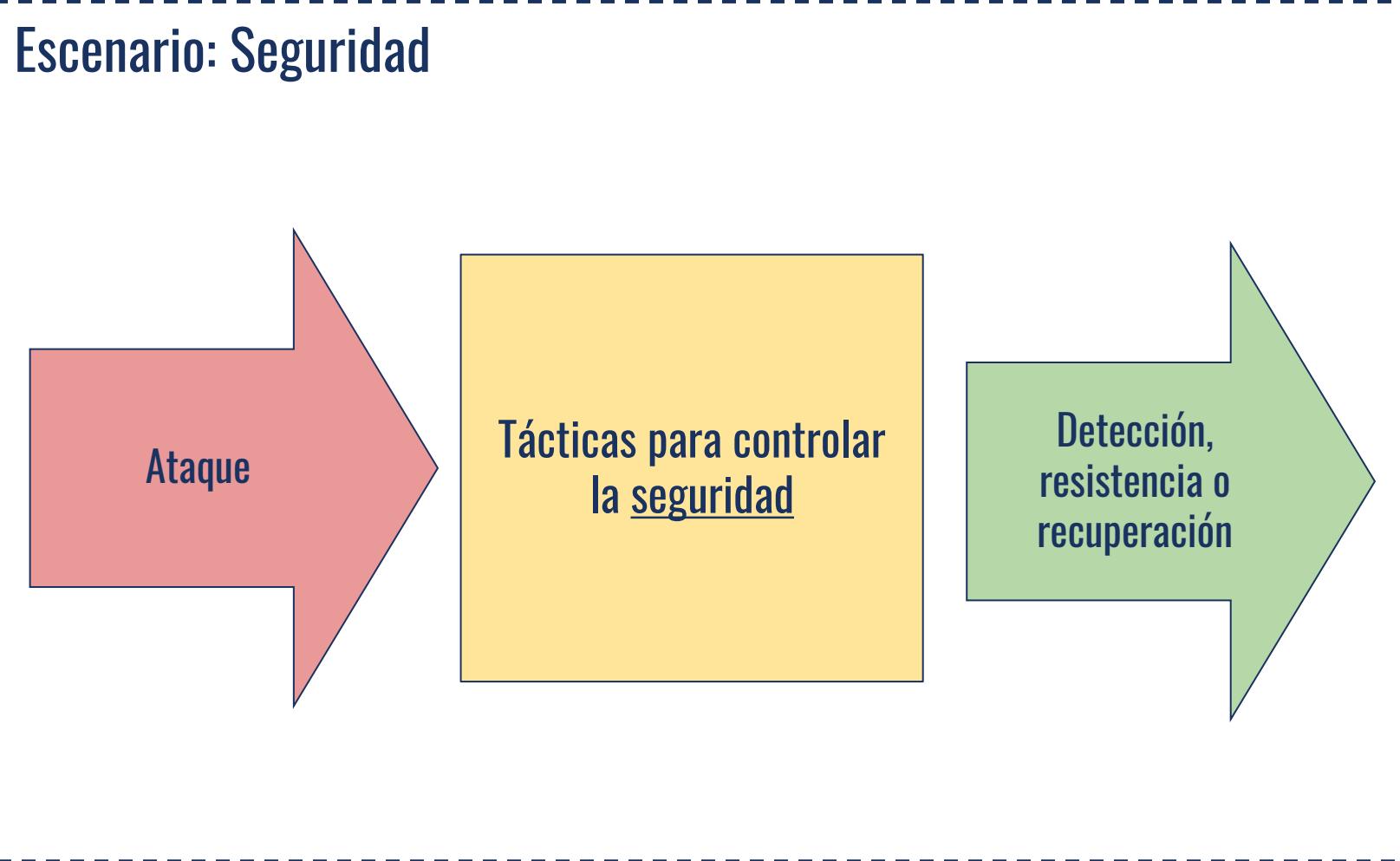


ESCENARIOS Y TÁCTICAS: EFICIENCIA DE EJECUCIÓN



ESCENARIOS Y TÁCTICAS: SEGURIDAD

Escenario: Seguridad



Ataque

Táticas para controlar
la seguridad

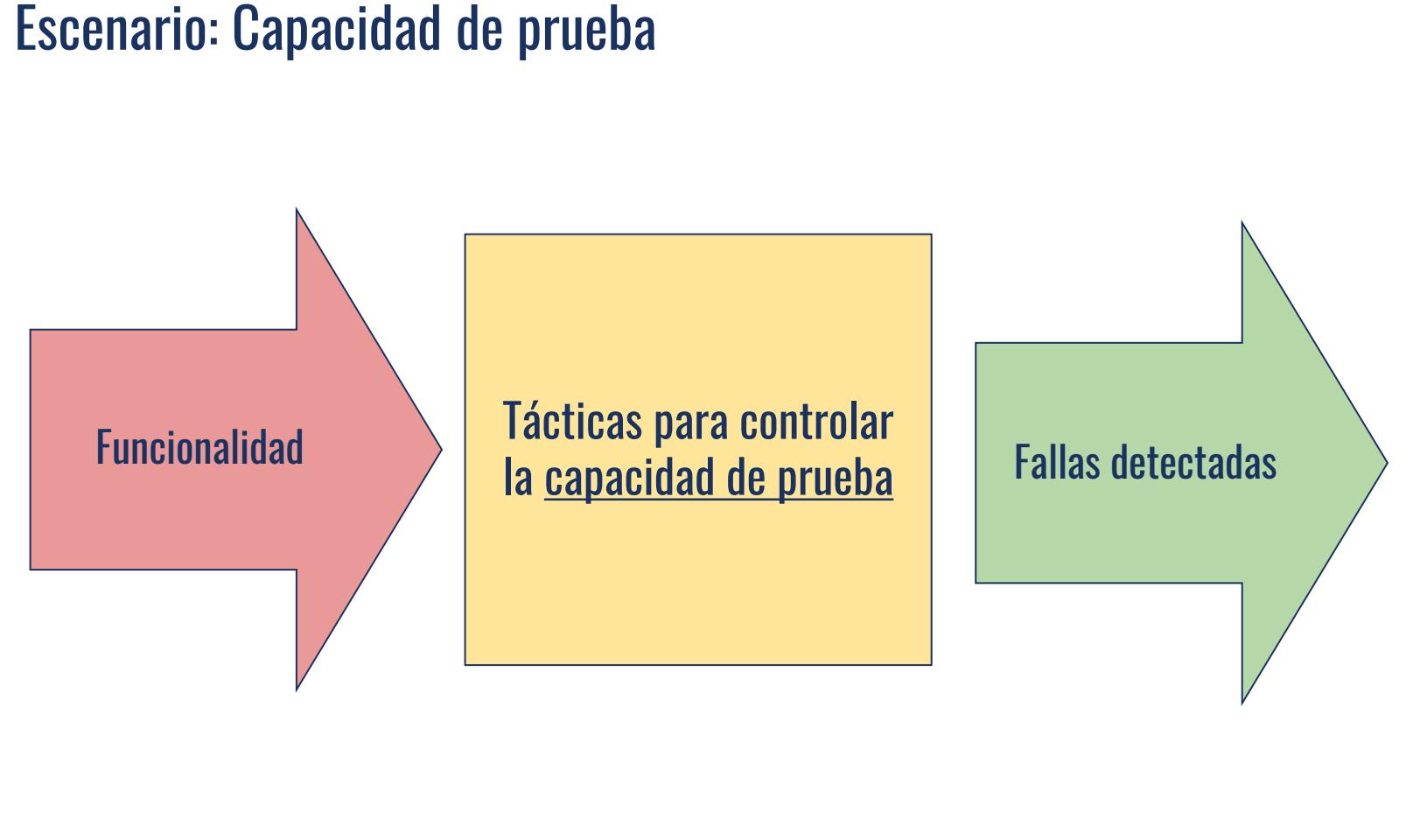
Detección,
resistencia o
recuperación

ESCENARIOS Y TÁCTICAS: SEGURIDAD



ESCENARIOS Y TÁCTICAS: CAPACIDAD DE PRUEBA

Escenario: Capacidad de prueba



Funcionalidad

Tácticas para controlar
la capacidad de prueba

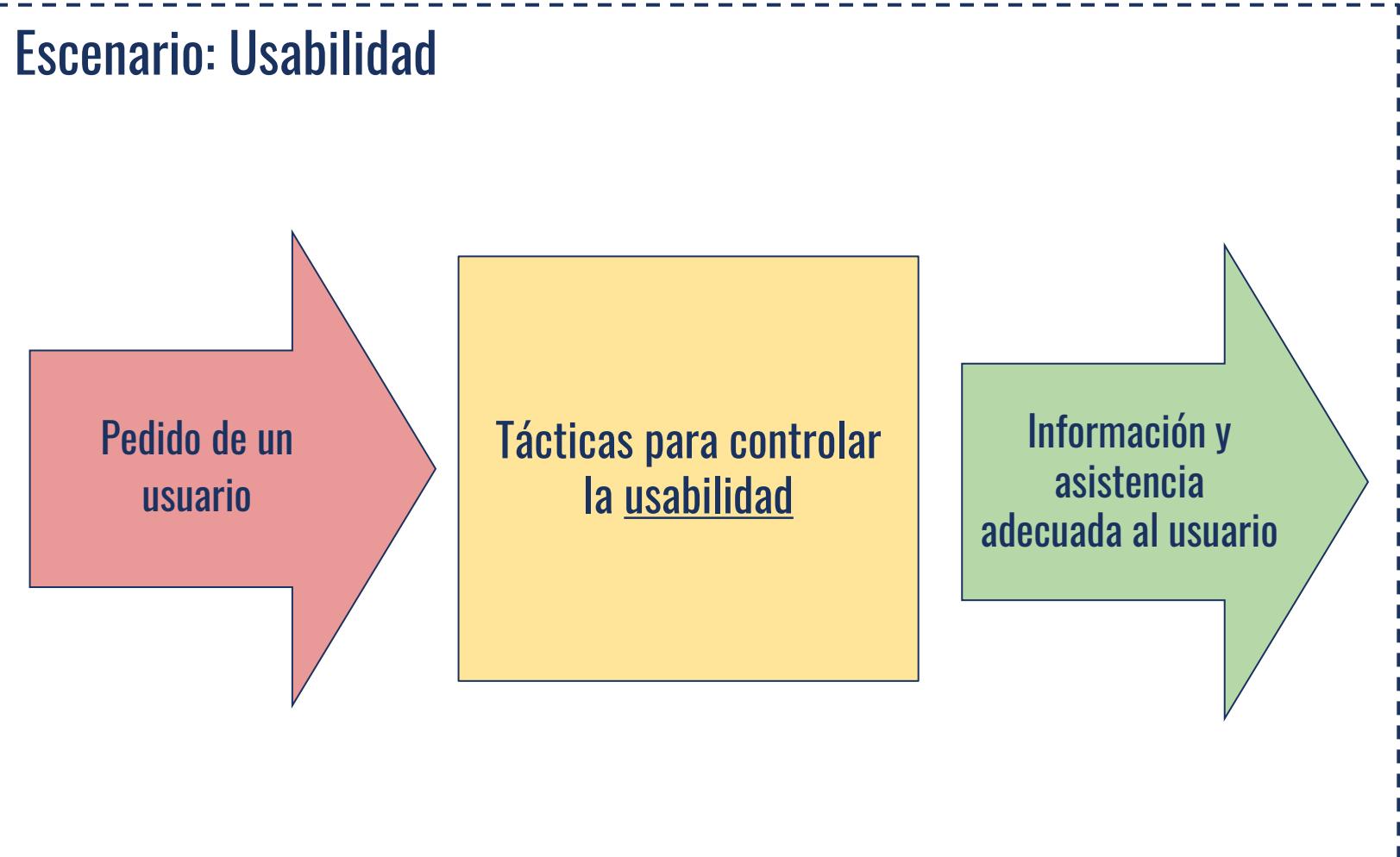
Fallas detectadas

ESCENARIOS Y TÁCTICAS: CAPACIDAD DE PRUEBA



ESCENARIOS Y TÁCTICAS: USABILIDAD

Escenario: Usabilidad



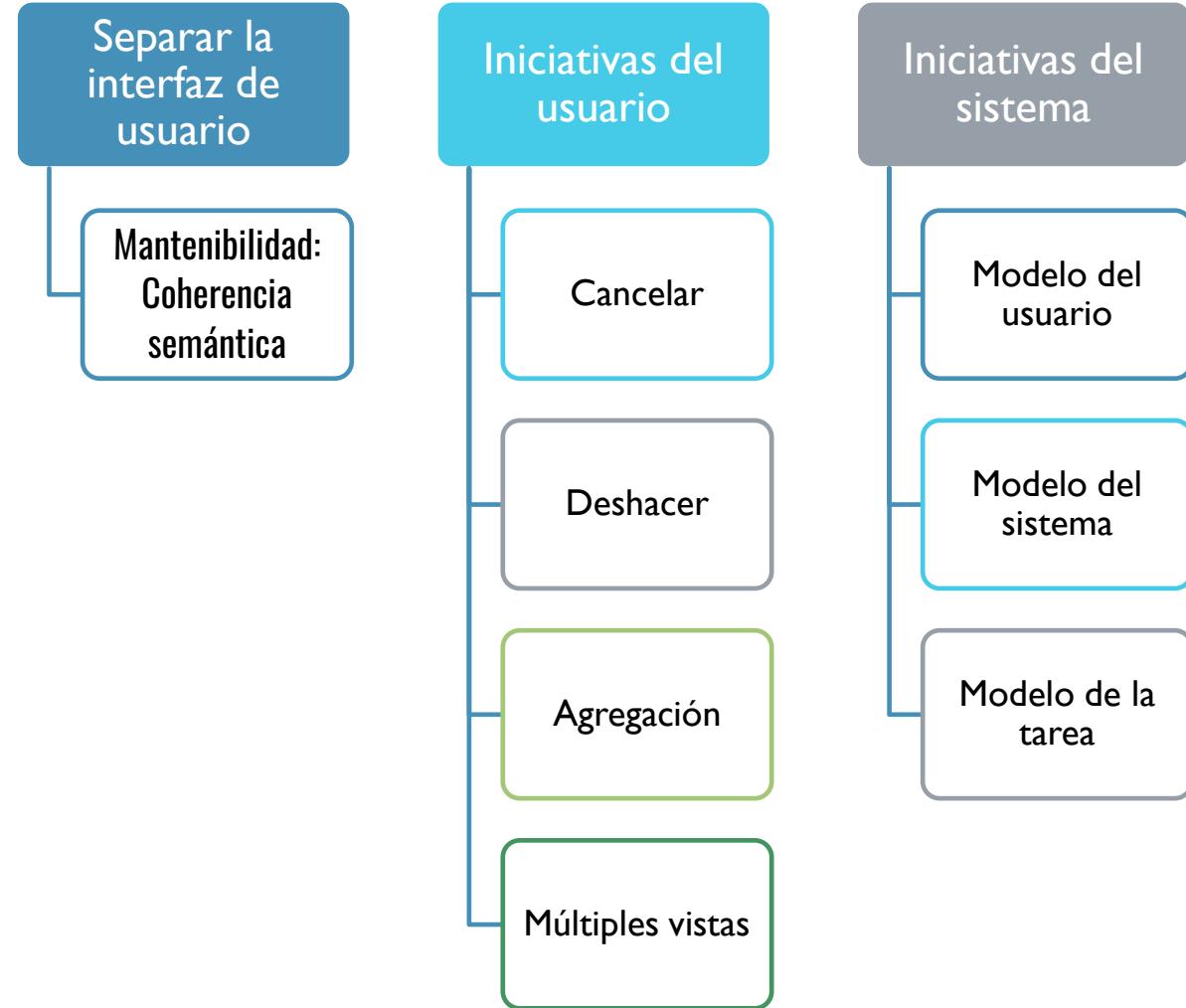
Pedido de un
usuario

Tácticas para controlar
la usabilidad

Información y
asistencia
adecuada al usuario

ESCENARIOS Y TÁCTICAS: USABILIDAD

Andrés Armando Sánchez Martín



CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: ALMACÉN Y FACTURA

- Diseñar una herramienta de gestión comercial para Windows. Factura en pesos, USD o Euro. Permite trabajar con lector de códigos de barras (EAN).
- Su arquitectura de 32 bits, mejora su rendimiento. Podrá trabajar de un modo más fácil y rápido, haciendo su empresa más operativa.
- Permite gestionar varios almacenes, un número indefinido de artículos a los cuales se les puede añadir en su ficha una foto del mismo, posibilidad de trabajar con proveedores, agentes, zonas, perfecto control de los vendedores, comisiones, descuentos, y depósitos. Y todo con un sencillo diseño.
- Incorpora mejoras en el tratamiento de inventarios, tarifas, cambios automáticos de precios, listados, informes, varios tipos de facturación, retenciones de I.R.P.F. remesa de recibos en soporte magnético, gestión de cobros, etc.
- El diseño de documentos es configurable por el usuario, incorporación de modelos estándar de documentos, restricción de accesos a la aplicación configurables por el administrador, enlace opcional con la contabilidad CONTAGEWIN, terminal punto de venta TPVWIN compartiendo la misma información, represente todo tipo de estadísticas de clientes, proveedores, agentes, zonas, etc

CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: ALMACÉN Y FACTURA

- Características:
 - Multi-almacén, amplio fichero de artículos, clientes, proveedores, comisiones, descuentos, depósitos.
 - Su diseño permite el enlace desde presupuestos/pedidos, pedidos/albaranes, albaranes/facturas, facturas/recibos.
 - Facturación en pesos, USD, Euro
 - Incorpora mejoras en tratamiento de inventarios, tarifas, cambios automáticos de precios, listados, informes, varios tipos de facturación, retenciones de IR.P.F, remesa de recibos en soporte magnéticos.
 - Gestión de cobros, envío de documentos vía fax desde la aplicación.

CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: DEIMOS

DEIMOS Space tiene como objetivo liderar el campo del desarrollo de software empotrado en tiempo real, que cubre el ciclo de vida completo de desarrollo y producción desde la fase de análisis de requisitos a través del diseño detallado y de arquitectura, codificación, testeo, integración, validación y aceptación. Estos desarrollos de software usan típicamente HOORA/UML para el análisis y modelado de requisitos, HRT-HOOD y Ada/C/C++ para el diseño e implementación.

El alcance de los desarrollos incluye software de aplicación de alto nivel así como servicios básicos de software empotrado para el tratamiento de telemetría y tele-comandos, programación de tareas, detección de fallos, aislamiento y recuperación y controladores de dispositivos.

Además de desarrollos para misiones específicas, DEIMOS Space realiza y participa en estudios tecnológicos dirigidos a mejorar el estado del arte del software empotrado en tiempo real.

- I. Software de control de vuelo: DEIMOS Space tiene capacidad para desarrollar software embarcado de control de vuelo critico en seguridad para: Sistemas de Control de órbita y Altitud (AOCS); Sistemas de Medida y Control de Altitud (ACMS); sistemas de Guiado, Navegación y Control (GNC). Se aplican técnicas de aseguramiento de productos software tales como análisis de Fiabilidad, Disponibilidad, Mantenibilidad y Seguridad (RAMS); Análisis del árbol de Fallos (FTA); Análisis de Criticidad y Efectos de Modo Fallo (FMECA); Análisis de Efectos de Errores Software (SEEA).

CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: DEIMOS

- 2. Software de tratamiento de Datos a Bordo: DEIMOS Space puede desarrollar software para subsistemas de Gestión de Datos a bordo (OBDH) que utiliza memoria de almacenamiento masivo, recogida y envío de telemetría, recepción y distribución de tele-comandos, gestión del interface de la plataforma de vuelo, y tareas de mantenimiento autónomo del sistema.
- 3. Software de control de Instrumentos: DEIMOS Space tiene capacidad para desarrollar software empotrado de tiempo real para ordenadores encargados de controlar instrumentos científicos a bordo de satélites y sondas espaciales. Típicamente el software realiza funciones específicas de comando y control de instrumentos, procesamiento de datos científicos y mantenimiento, control y monitorización del interface del instrumento y gestión de modos.
- 4. Validación de software de Segmento Espacio: DEIMOS Space puede realizar validación de software usando técnicas en línea con el estado del arte y entornos de testeo, e incluyendo métodos de análisis estático y dinámico.
- 5. Simulación de Sistemas Embarcados: DEIMOS Space complementa sus actividades en el rea de software embarcado con experiencia en simulación como medio para apoyar el desarrollo, validación y análisis de productos software. Para mas información acerca de las actividades de DEIMOS Space en el campo de Sistemas en Tiempo Real, por favor contacte con D. Mike Rennie.

BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué compone una arquitectura?
- ¿Cuáles son los tipos de ambientes del software?
- ¿Cuáles son los patrones y estilos de arquitectura mas usados en la actualidad?

Taller:

- Entregar Documento con la identificación de los Atributos de Calidad, Escenarios de Calidad y Los Acuerdos para cada Caso de Estudio.
- Entrega clase 4.2





(1306)

ARQUITECTURA DE SOFTWARE

4.1 Estilos y Patrones Arquitecturales |

Agenda

- Tipos de ambientes para el software
- Arquitectura
- Componentes y conectores
- Patrones y Estilos Arquitecturales



TIPOS DE AMBIENTES PARA EL SOFTWARE: ECOSISTEMAS

Red Sockets

Terminal

Menú de Interacción

Escritorio/Pantalla Grafica

- Mouse
- Táctil
- Mandos

Web

HCI

- Voz
- Gestos
- Imagen

Realidad Aumentada

Máquinas Industriales M2M

Vehículos

TIPOS DE AMBIENTES PARA EL SOFTWARE

Sistemas Locales

- PC
- Dispositivos
- Móviles

Sistemas en red

- Orientados a Conexión
- No Orientados a Conexión



Pontificia Universidad
JAVERIANA
Bogotá

Andrés Armando Sánchez Martín

TIPOS DE AMBIENTES PARA EL SOFTWARE: LOCALES

Aplicaciones
de Escritorio

Procesos
Bach

Aplicaciones
Portables

Sistemas
Operativos

Videojuegos

Aplicaciones
de
Configuración



Pontificia Universidad
JAVERIANA
Bogotá

Andrés Armando Sánchez Martín

TIPOS DE AMBIENTES PARA EL SOFTWARE: EN RED

Aplicaciones
Web

PWA

Web Apps

Middleware

Servicios

Computación
Móvil



TIPOS DE AMBIENTES PARA EL SOFTWARE: DESPLIEGUE DE SOFTWARE

Mainframes

Servidores

PC

Dispositivos

Maquinas
Virtuales

Servidores
Web

Servidores de
Aplicaciones

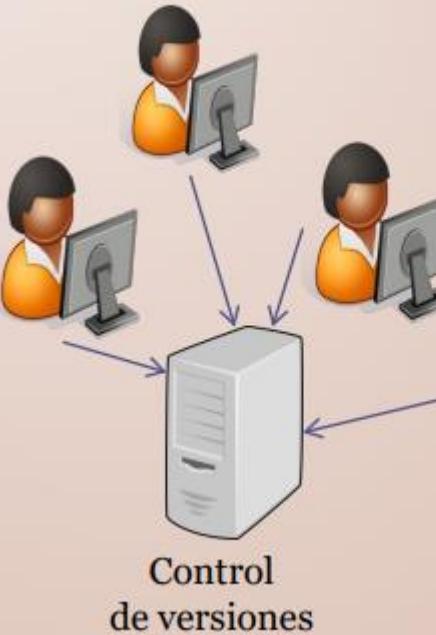
Interpretes

Contenedores

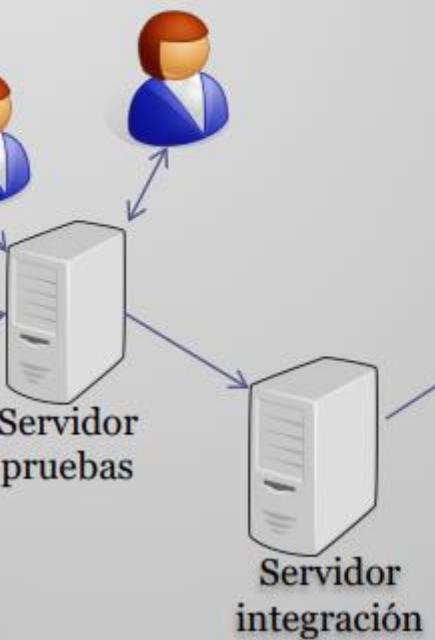
Cloud

TIPOS DE AMBIENTES PARA EL SOFTWARE: ENTORNOS

Entorno de desarrollo



Entorno de pruebas



Entorno de producción



Servidor producción
Granja servidores

Entorno de ensayo (staging) también se utiliza en ocasiones

ARQUITECTURA

- 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

COMPONENTES Y CONECTORES

Componentes

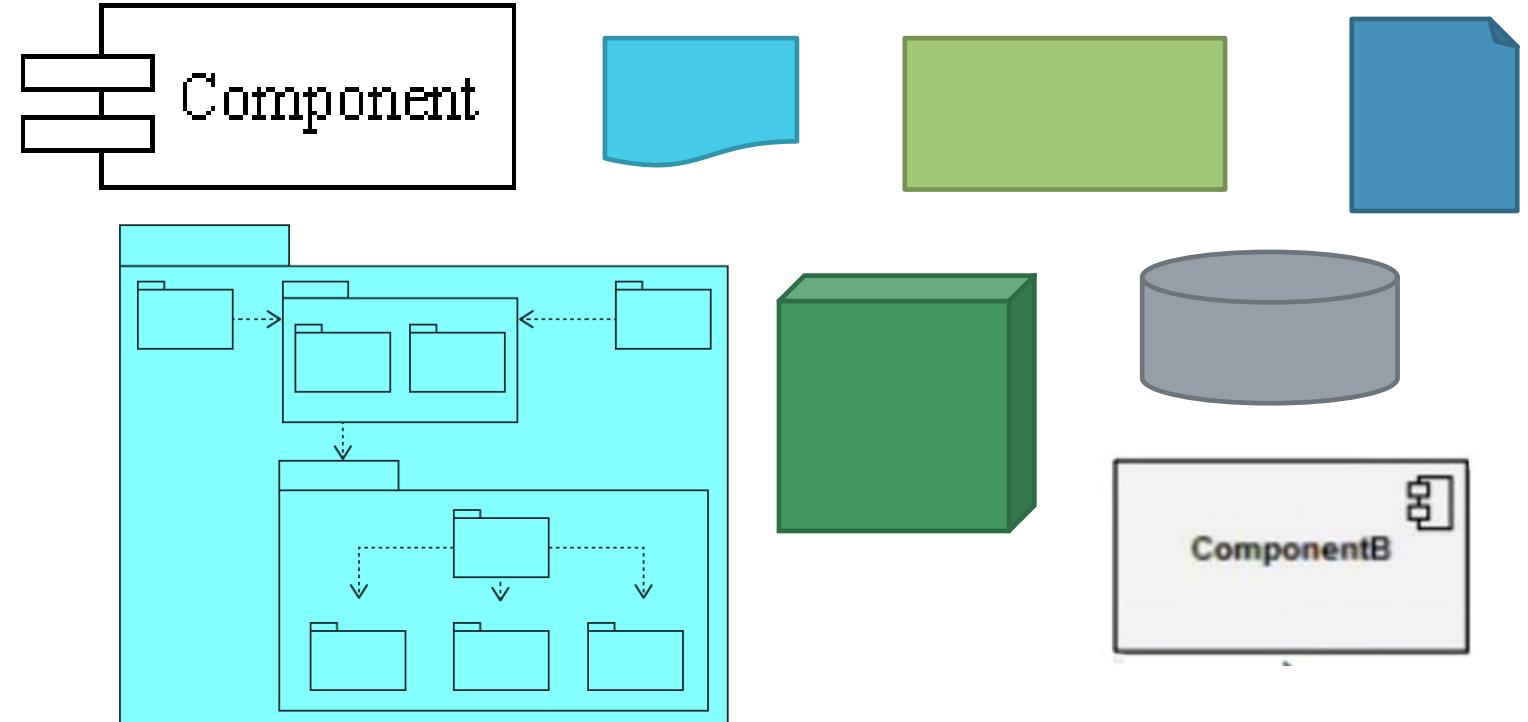
- Son partes de nuestro sistema que cumplen una función específica. Estos mismos componentes “modulares” pueden estar formados por más componentes, ya bien objetos o capas, que actúan como subcomponentes en su interior.
- La comunicación existente entre ellos y se lleva a cabo por medio de conectores.

Conectores

- Hacen de comunicación entre los componentes. Estos no se limitan a un problema específico, pudiendo ser analizados de forma separada de los problemas a resolver.
- Estos no están asociados a un dominio específico y son independientes a la hora de su análisis, pudiendo un e-commerce o una red social el mismo tipo de conector.

COMPONENTES Y CONECTORES: COMPONENTES

- Representan elementos del dominio y contexto
- Dependiendo el nivel de abstracción pueden representar:
 - Funcionalidades
 - Sistemas
 - Productos de software
 - Dispositivos
 - Datos
- Pueden ser:
 - Componentes
 - Subcomponentes



COMPONENTES Y CONECTORES: CONECTORES

Llamado a procedimiento:

- Invocan de un componente a otro componente y esperan una respuesta.
- Realizan la invocación de un componente a otro y esperan una respuesta

Enlace:

- Vinculan fuertemente un componente a otro, incluso para la compilación. Visto en lenguajes compilados, y en componentes que forman parte de un monolito
- Vinculan fuertemente un componente con otro, haciéndolo necesario para la compilación.

Evento:

- Permiten a un componente notificar un evento (que algo sucedió), y a otros componentes escuchar y reaccionar ante un evento.
- Permiten la escucha y notificación entre componentes (Listener)

Adaptador:

- Ayudan a compatibilizar la interfaz de un componente con la de otro componente
- Permite conectar/compatibilizar la interfaz de un comp. con la de otro.

COMPONENTES Y CONECTORES: CONECTORES

Acceso a datos:

- Nos ayudan a acceder a recursos compartidos de datos, como APIs, sistemas de archivos y bases de datos. Compatibiliza la interfaz del dato con la interfaz que espera el componente que estamos usando.
- Permite el acceso a recursos compartidos (db, archivos, apis, etc)

Flujo:

- Permite la recolección de datos en un flujo de información continuo por parte de otro componente que tiene intereses en obtener varios o todos los datos del flujo.
- Permite el acceso a datos que están en constante flujo

Arbitraje:

- Permite la coordinación entre componentes en acceso, trabajo, etc.
- Coordinan los permisos de acceso a un recurso entre componentes y deciden quien se encarga de distribuir dichos comportamientos.
- Ej: Test A/B, teniendo varios componentes disponibles recibimos un pedido y se decide qué versión enviar para comparar diferentes atributos de calidad.

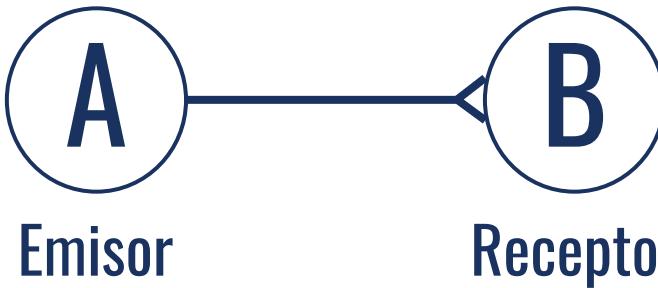
Distribuidor:

- Facilita la distribución de un mensaje a varios componentes a través de un solo conector.
- Permite distribuir información de un componente a varios otros.

CONECTORES: LLAMADOS

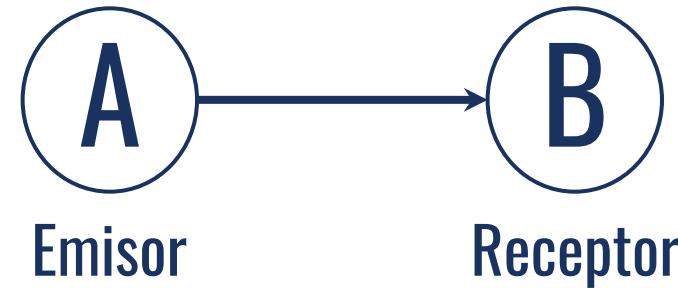
Síncrono

- Emisor envía una solicitud a otro y espera la respuesta para seguir su ejecución. Delegando así el procesamiento.



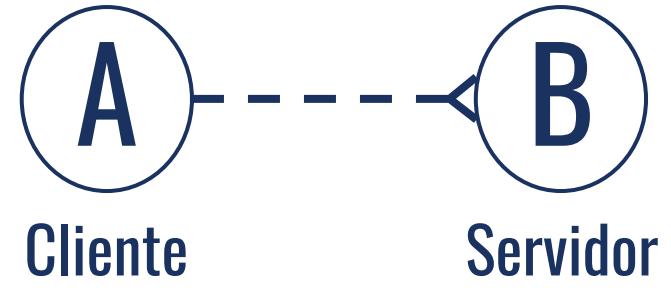
Asíncrono

- Un componente llama a otro sin detener su ejecución y evaluando la respuesta en algún momento determinado.



Cliente - Servidor

- Comunicación desde el cliente al servidor, cliente puede no saber quien es el servidor.



CONECTORES: CONEXIÓN

Enrutador

- Facilita conexión entre emisor y un grupo de receptores identificados

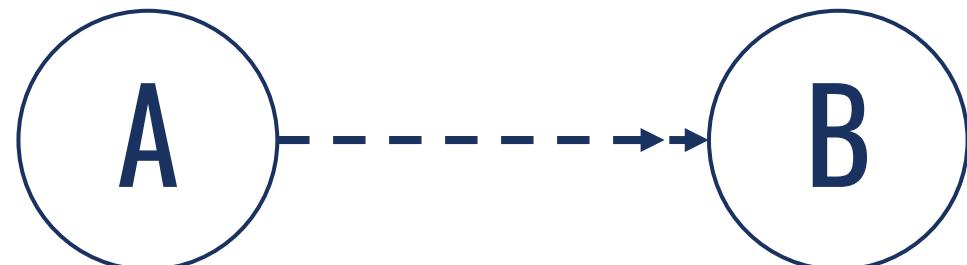


Emisor

Receptor

Difusión

- Difunde mensaje a todos los receptores identificados

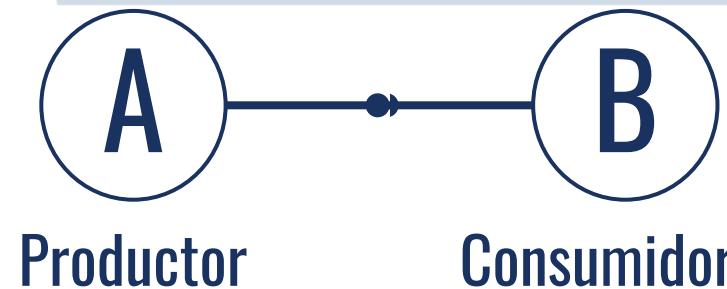


Emisor

Receptor

CONECTORES: ESTRUCTURA

Andrés Armando Sánchez Martín

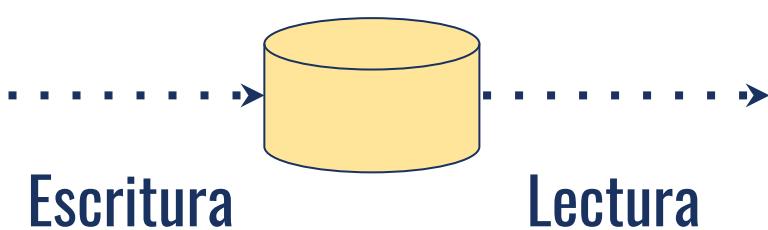


Cola

- Busca compatibilizar la velocidad del productor con la del consumidor

Pizarra o Repositorio

- Orientado a leer y/o escribir datos. Buscan garantizar estos procesos.



Publicar - subscribir

- Permite enviar mensajes de eventos y que otros componentes se suscriban sin tener que conocerse los componentes.



PATRONES Y ESTILOS ARQUITECTURALES: ESTILOS

- ¿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/contras de un estilo
 - Valorar las consecuencias de desviarse del estilo



PATRONES Y ESTILOS ARQUITECTURALES: PATRÓN

- 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



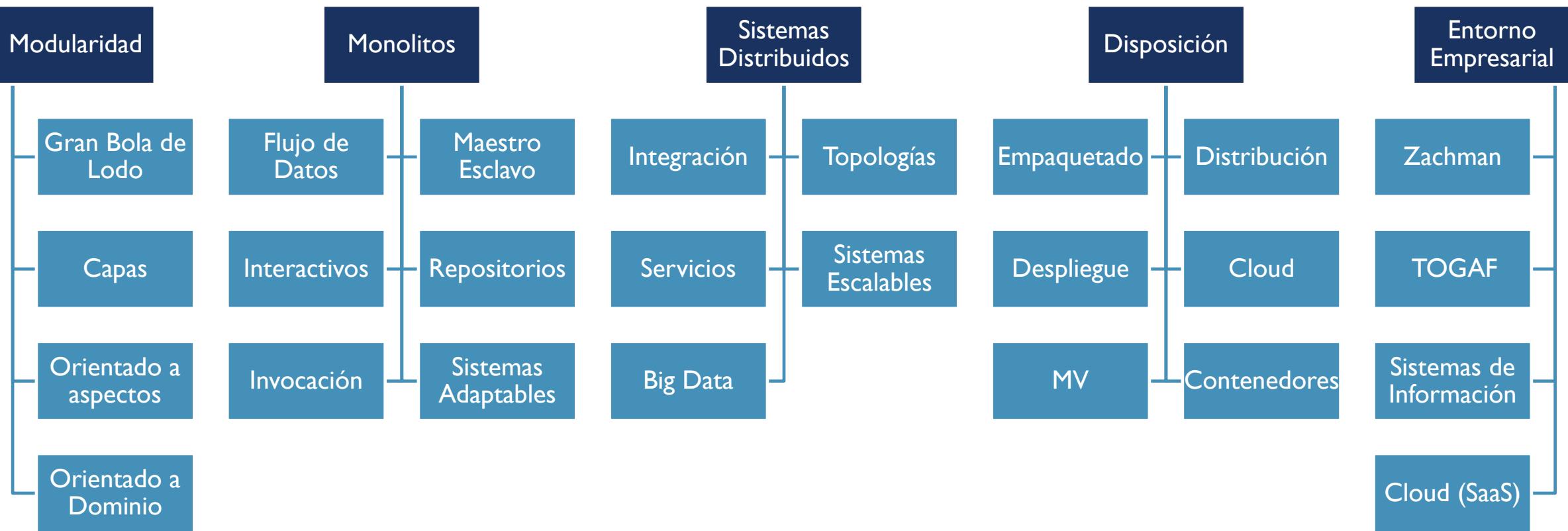
PATRÓN VS ESTILO

- 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)
- 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



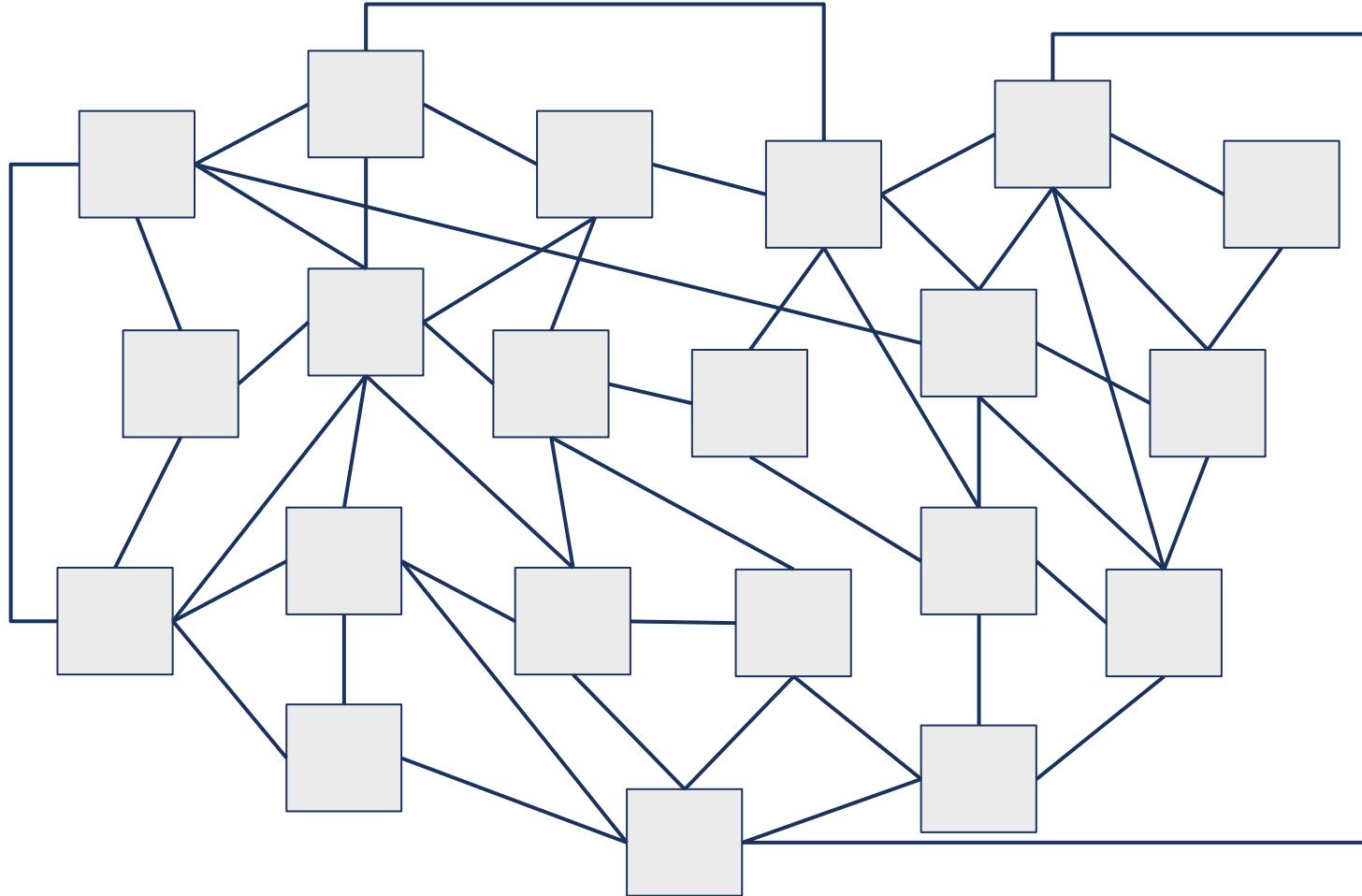
CLASIFICACIÓN DE ARQUITECTURAS

Andrés Armando Sánchez Martín



PATRONES ARQUITECTURALES: GRAN BOLA DE LODO / BIG BALL OF MUD

Andrés Armando Sánchez Martín



PATRONES ARQUITECTURALES: MVC

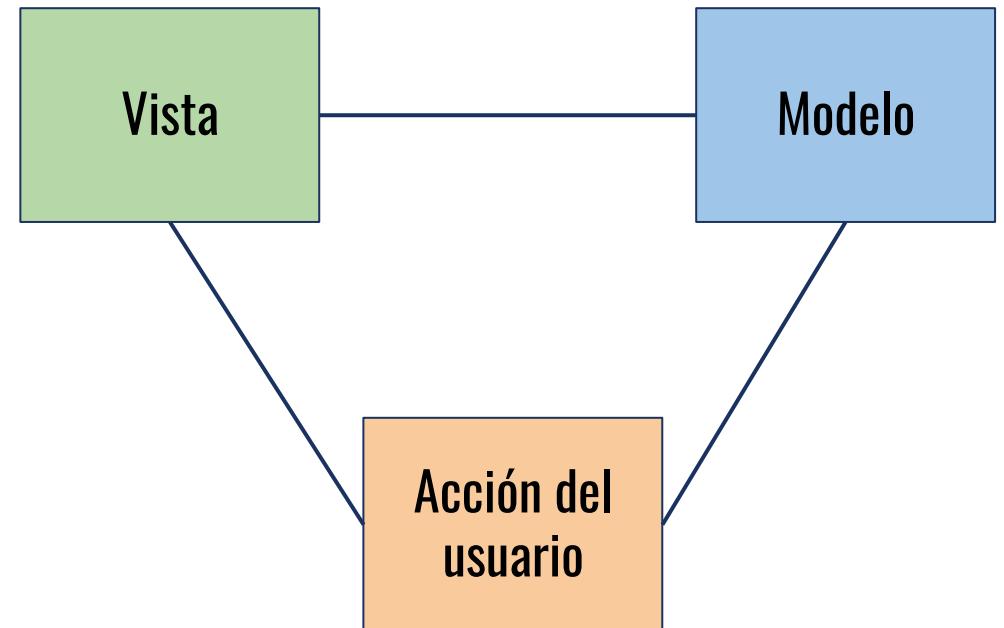
Andrés Armando Sánchez Martín

Modelo Vista Controlador / *Model View Controller*

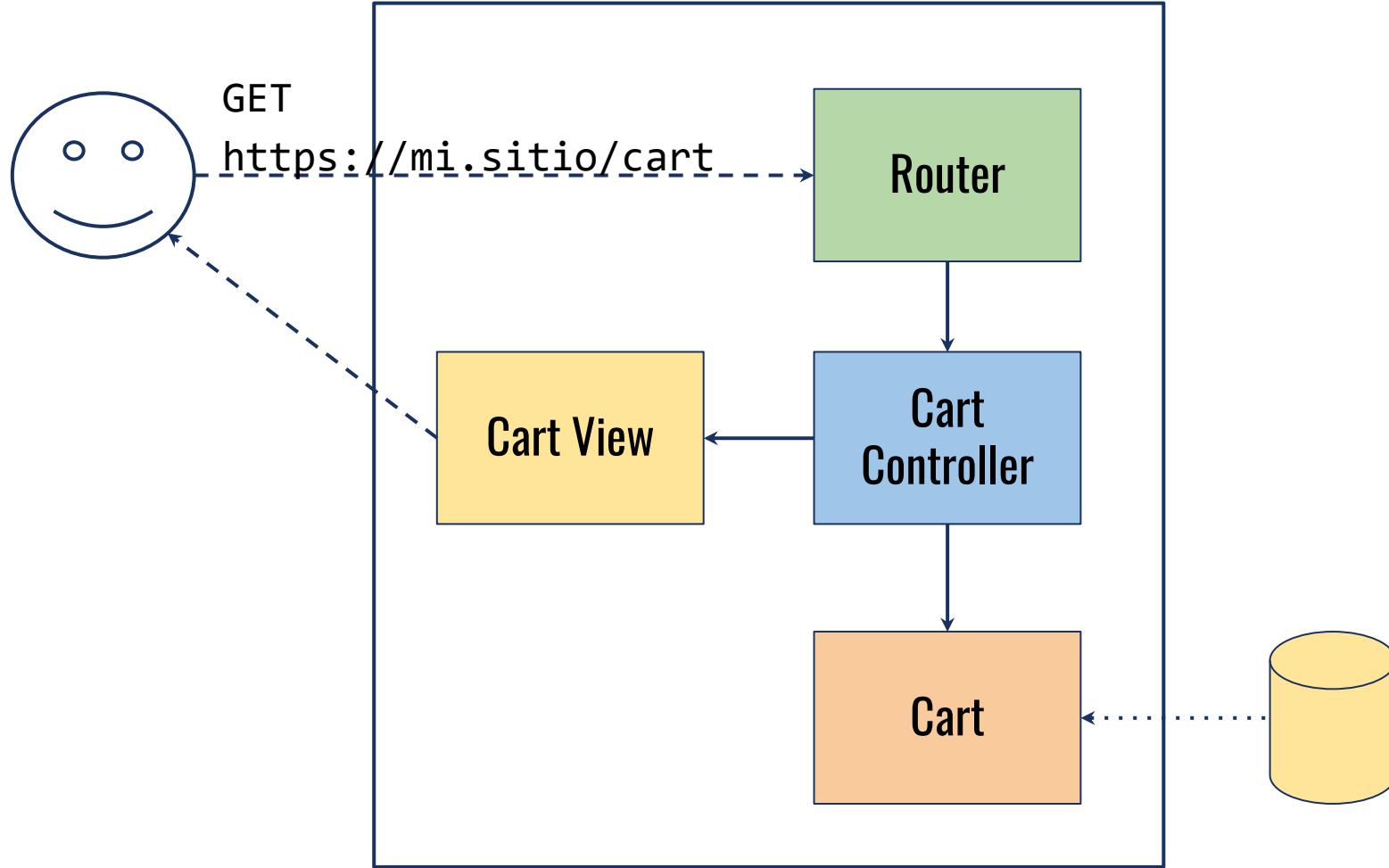
Modelo Vista Modelo-de-Vista / *Model View ViewModel*

Modelo Vista Presentador / *Model View Presenter*

Flux / *Flux*

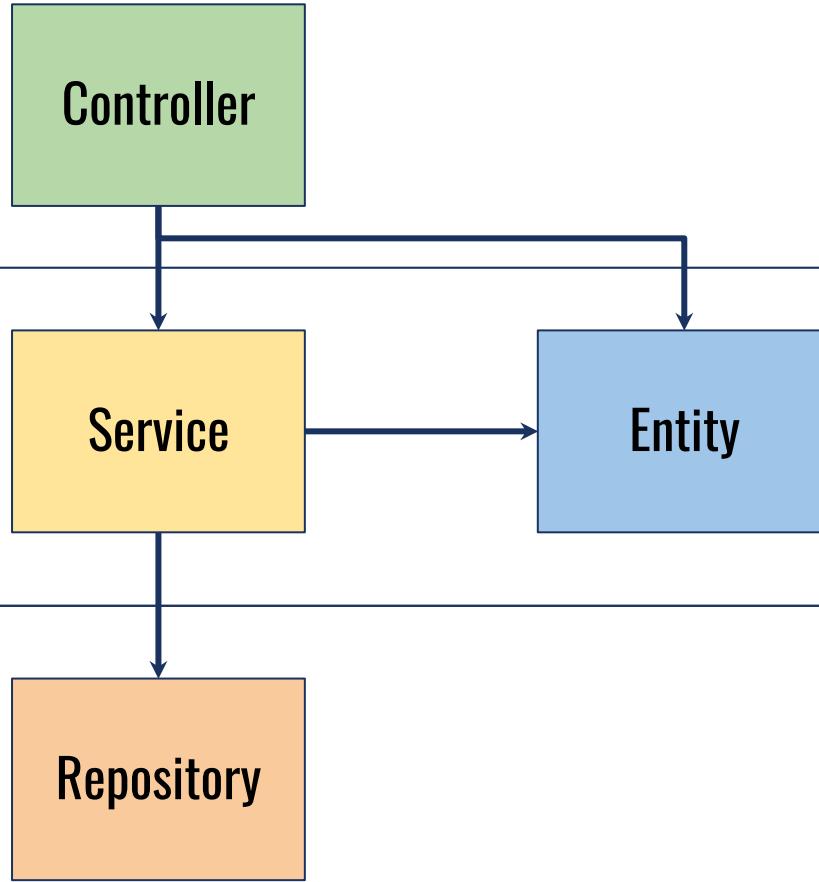


PATRONES ARQUITECTURALES: MVC



PATRONES ARQUITECTURALES: CAPAS / LAYERED

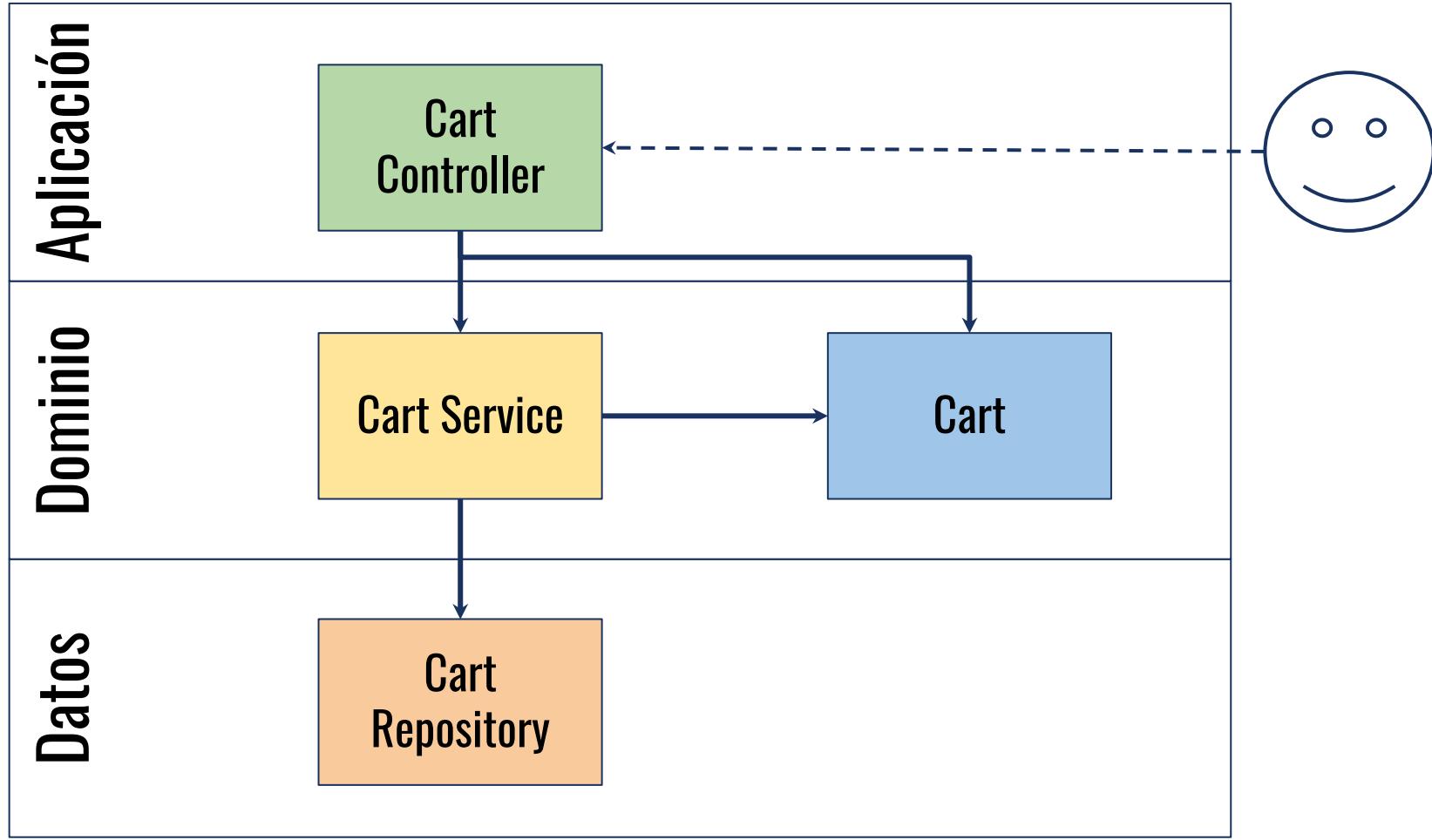
Aplicación
Dominio
Datos



PATRONES ARQUITECTURALES: CAPAS / LAYERED

Andrés Armando Sánchez Martín

25



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

Andrés Armando Sánchez Martín

- ¿Qué otros patrones de arquitectura?
- ¿Qué estilos de arquitectura?





(1306)

ARQUITECTURA DE SOFTWARE

4.2 Estilos y Patrones Arquitecturales 2

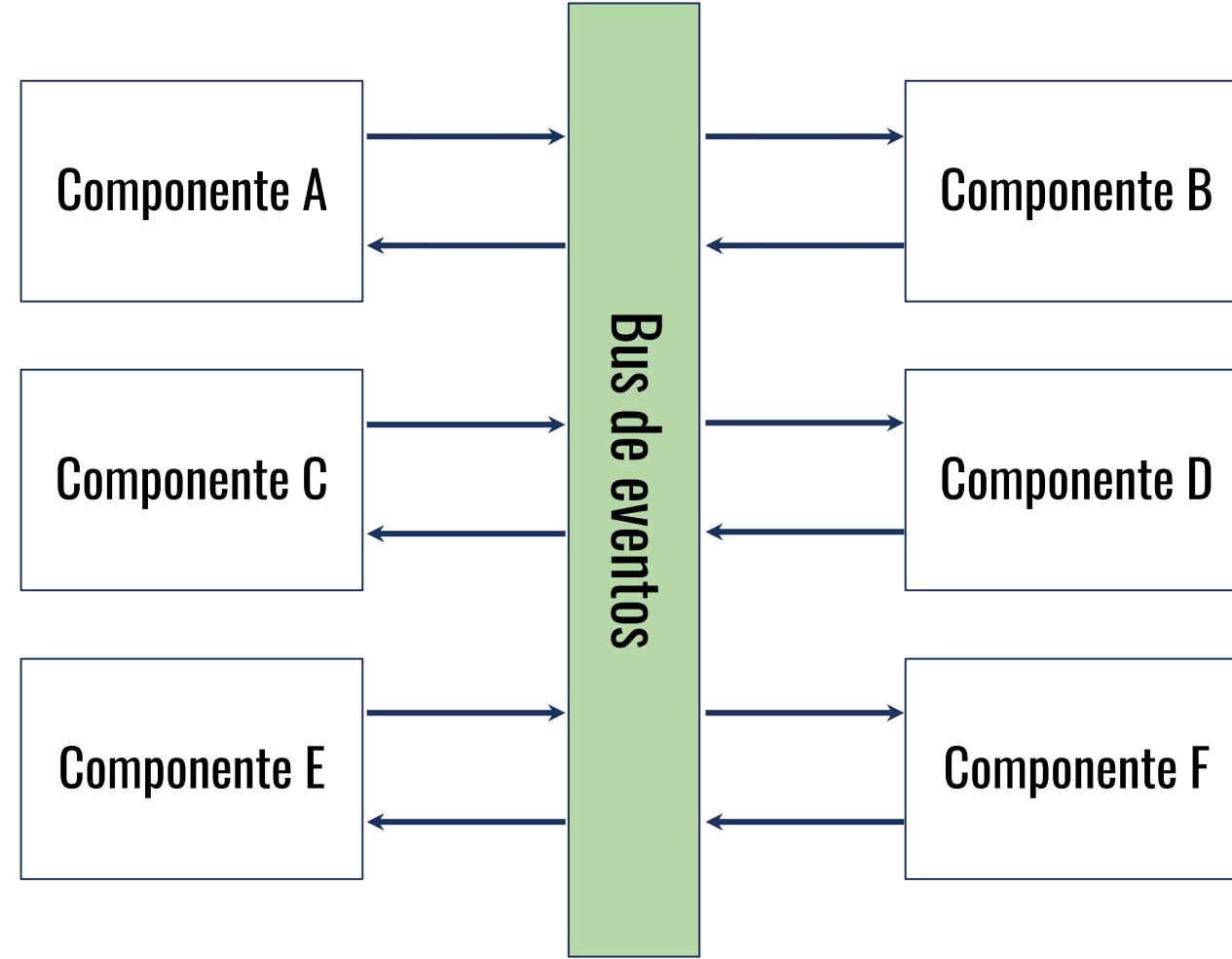
Agenda

Andrés Armando Sánchez Martín

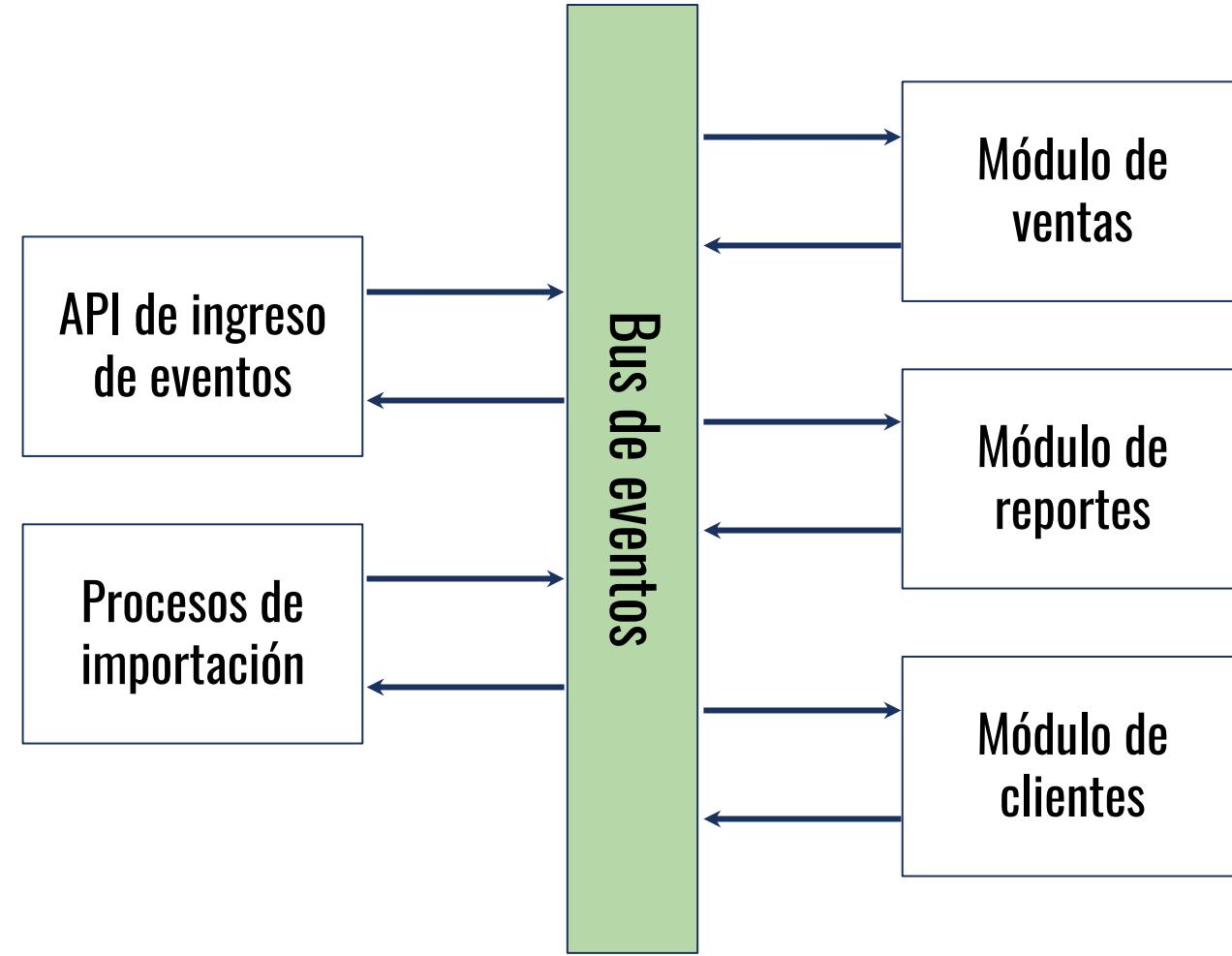
- Patrones Arquitecturales
- Estilos Arquitecturales



PATRONES ARQUITECTURALES: ORIENTADO A EVENTOS / EVENT-DRIVEN



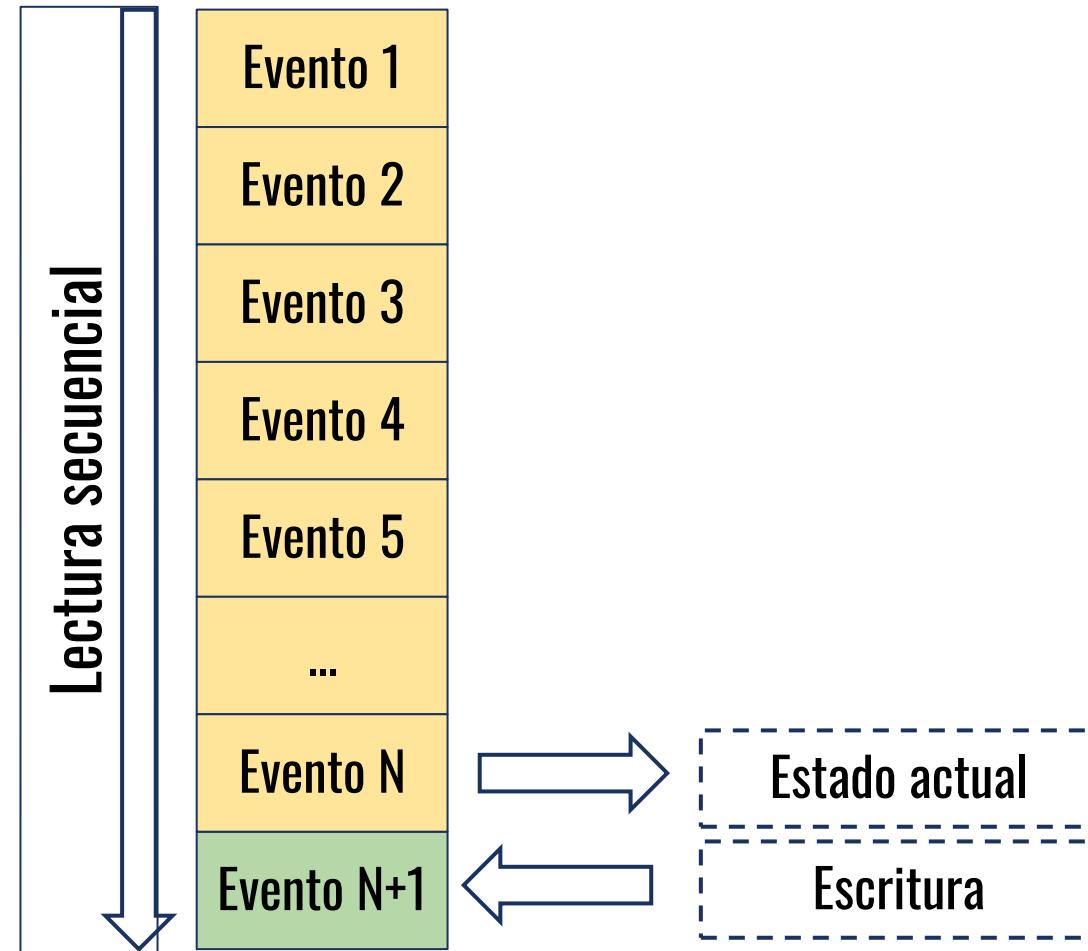
PATRONES ARQUITECTURALES: ORIENTADO A EVENTOS / EVENT-DRIVEN



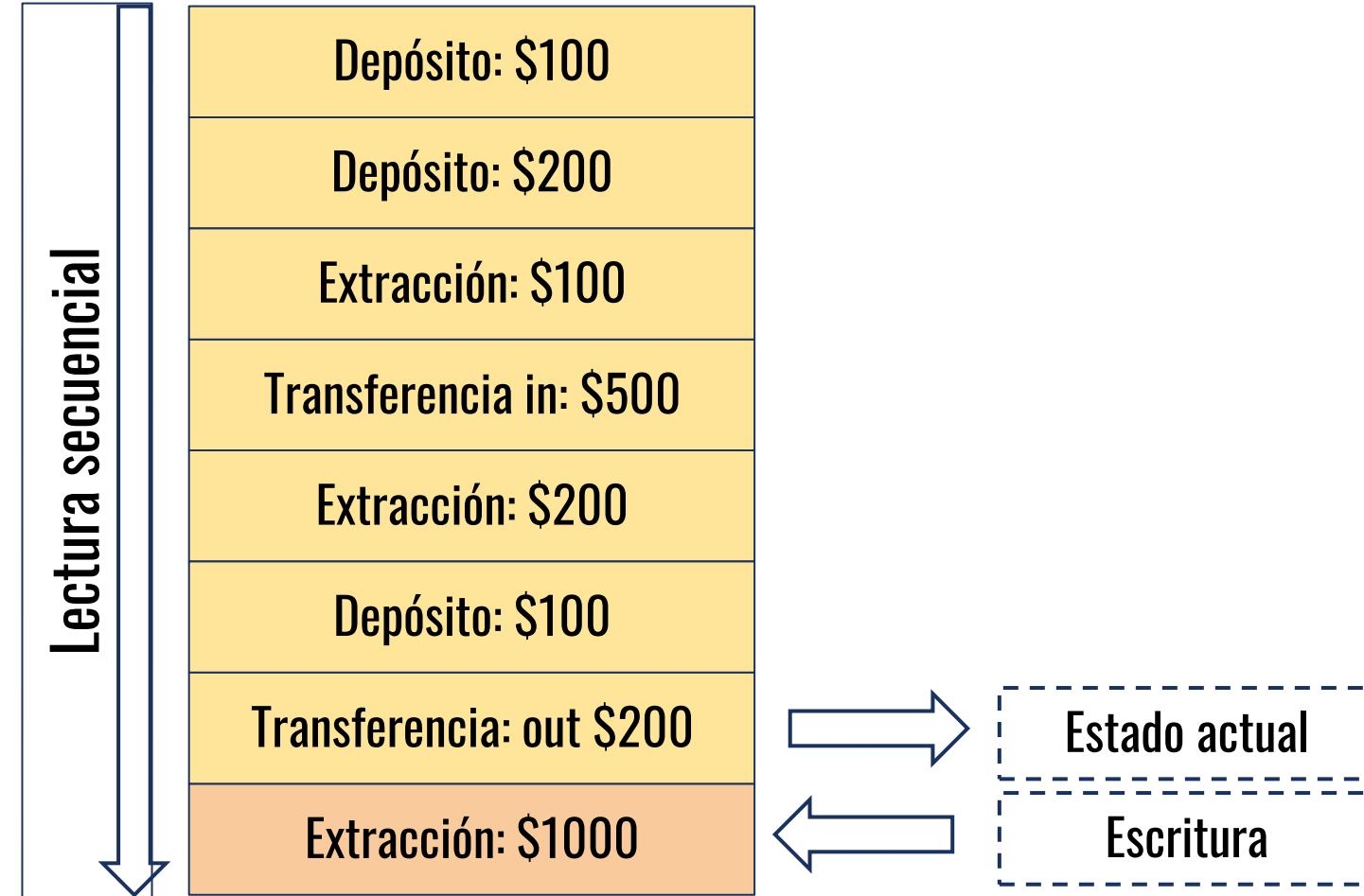
PATRONES ARQUITECTURALES: PROVISIÓN DE EVENTOS / EVENT SOURCING

Andrés Armando Sánchez Martín

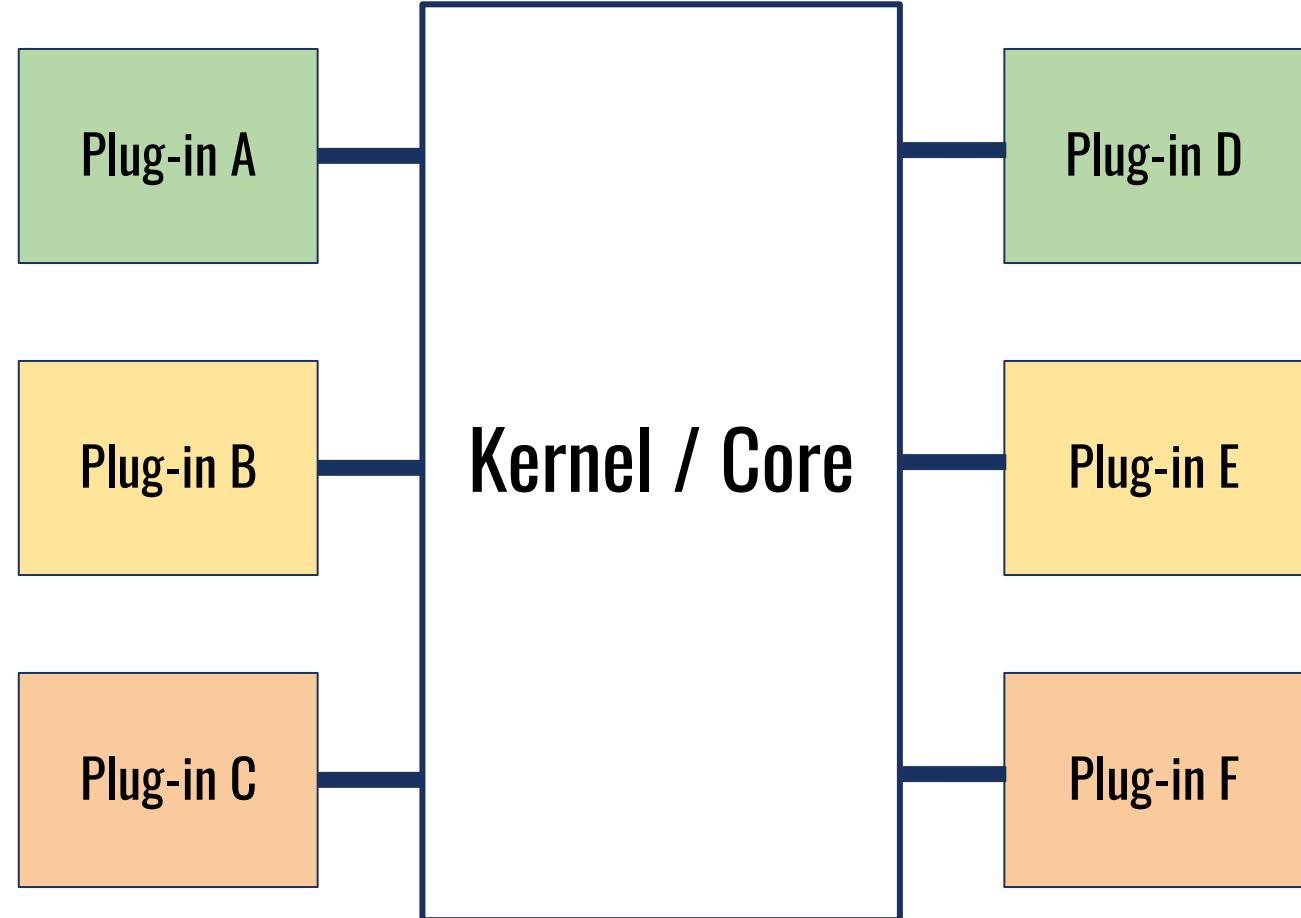
5



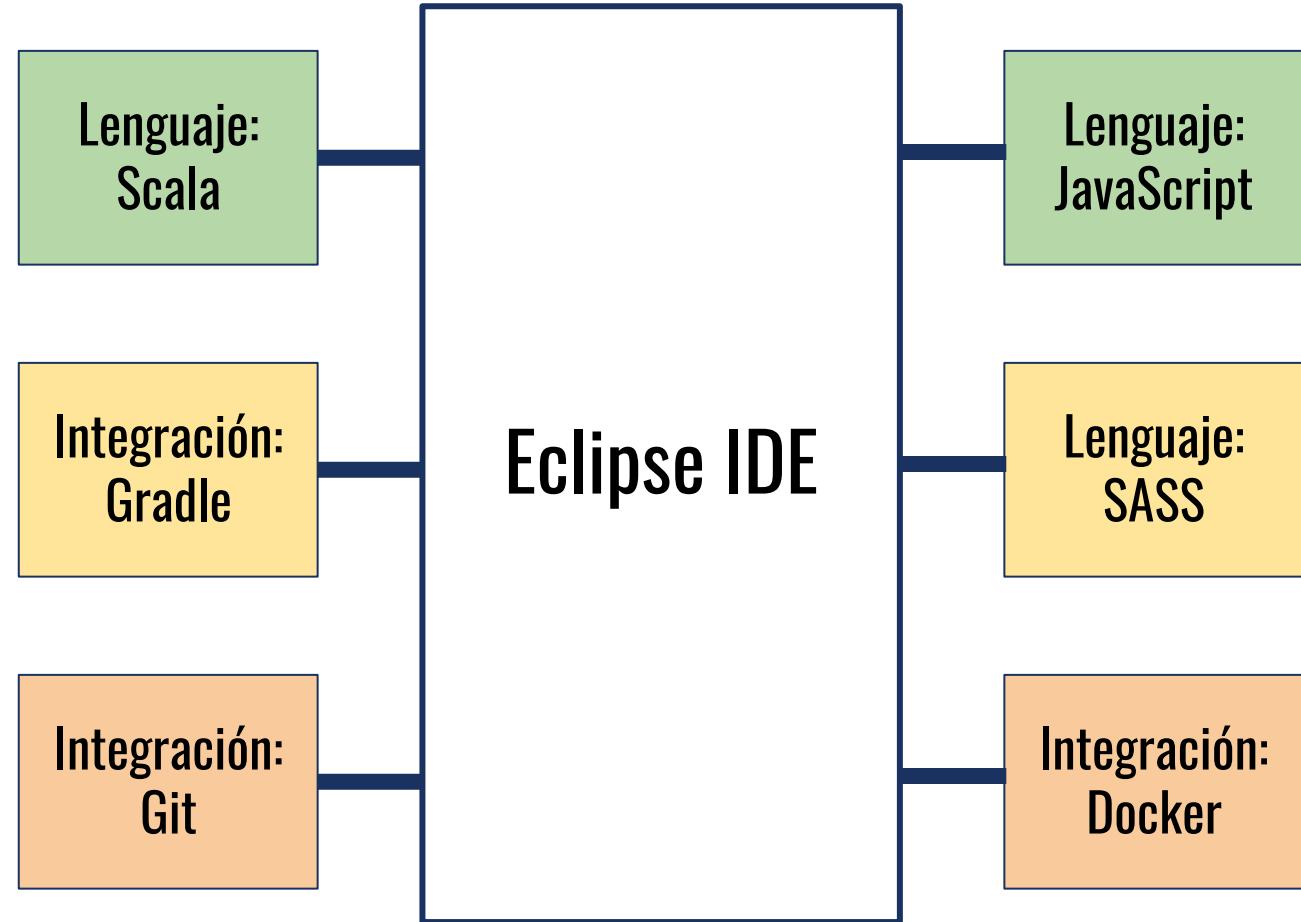
PATRONES ARQUITECTURALES: PROVISIÓN DE EVENTOS / EVENT Sourcing



PATRONES ARQUITECTURALES: MICROKERNEL - PLUG-IN'S

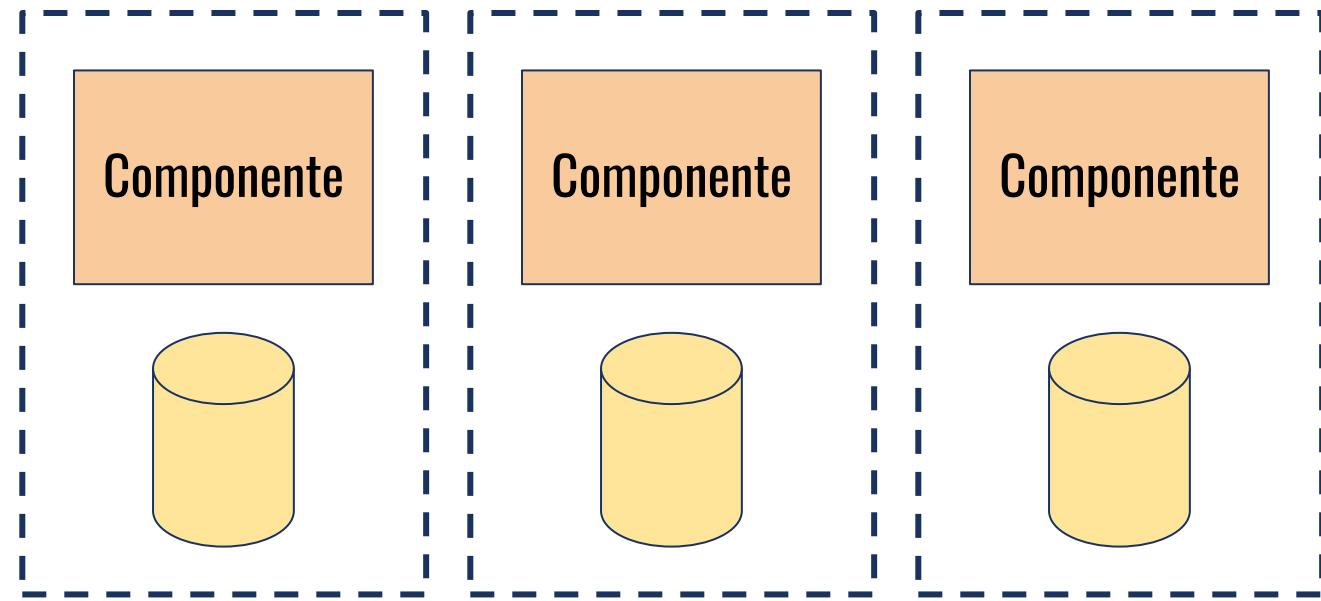


PATRONES ARQUITECTURALES: MICROKERNEL - PLUG-IN'S



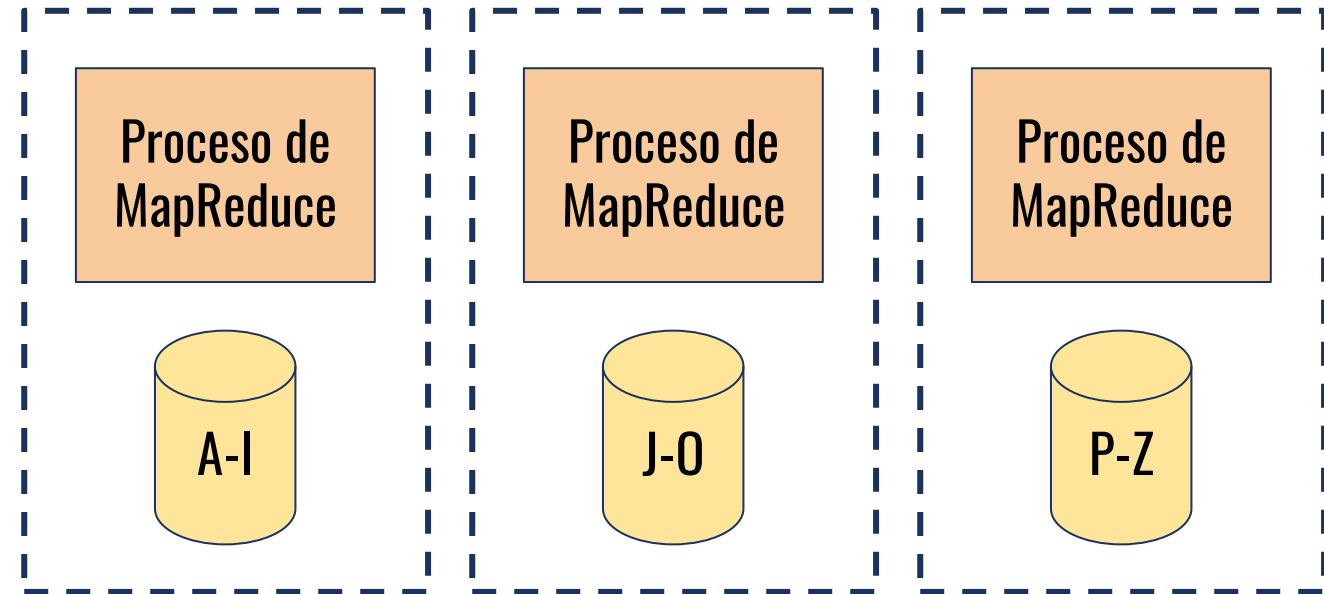
PATRONES ARQUITECTURALES: COMPARTE-NADA / SHARED-NOTHING

Andrés Armando Sánchez Martín

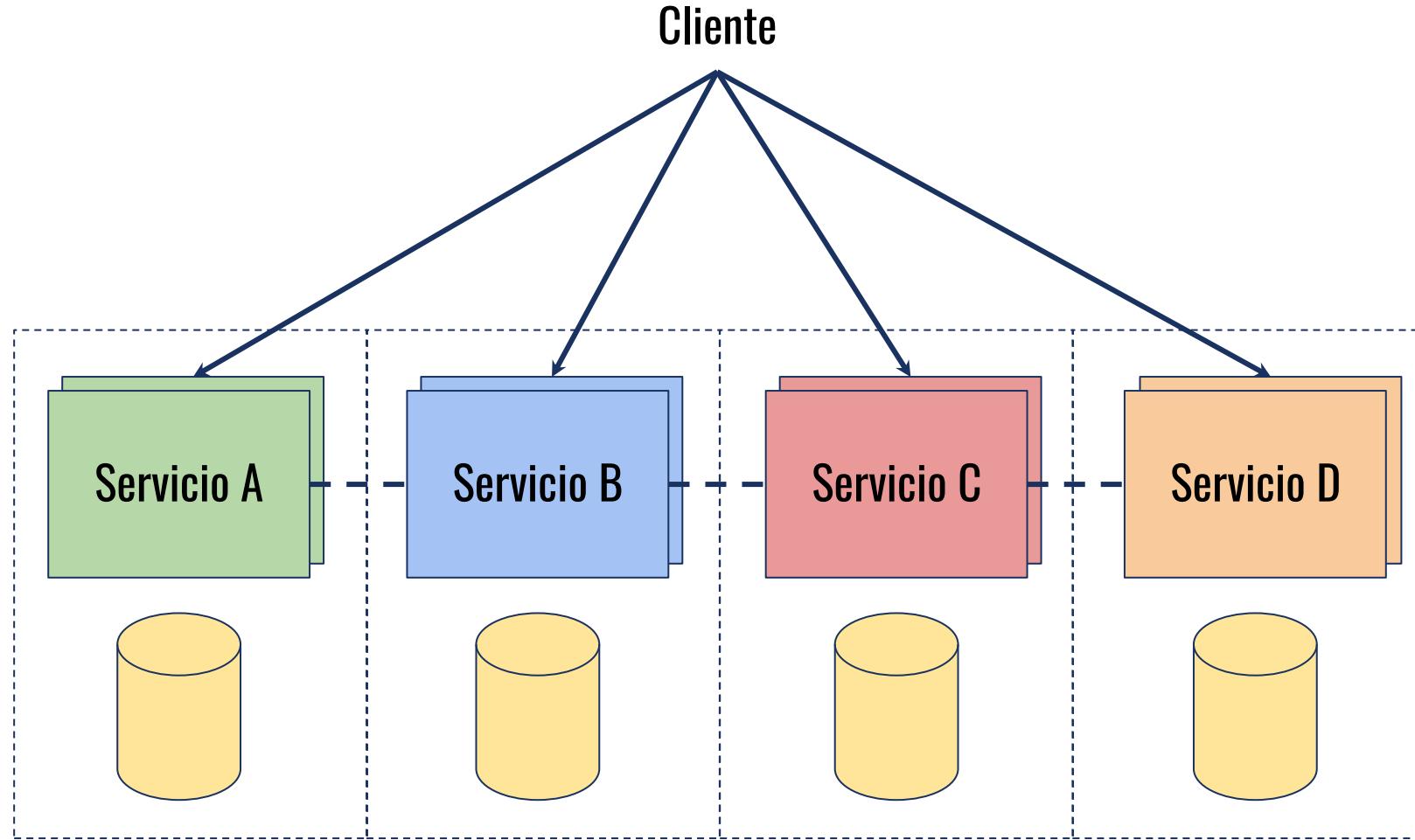


PATRONES ARQUITECTURALES: COMPARTE-NADA / SHARED-NOTHING

Andrés Armando Sánchez Martín

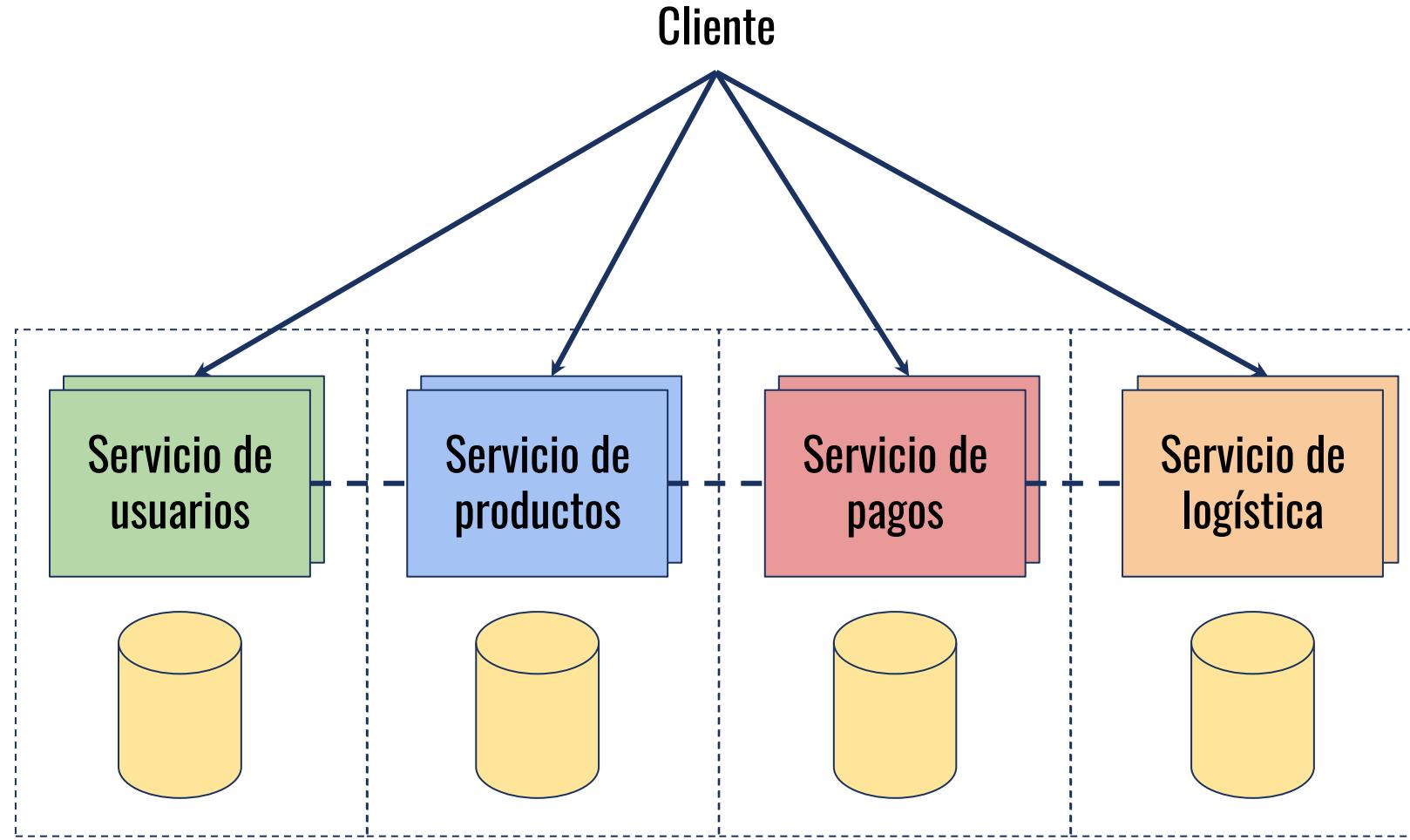


PATRONES ARQUITECTURALES: MICROSERVICIOS / MICROSERVICES

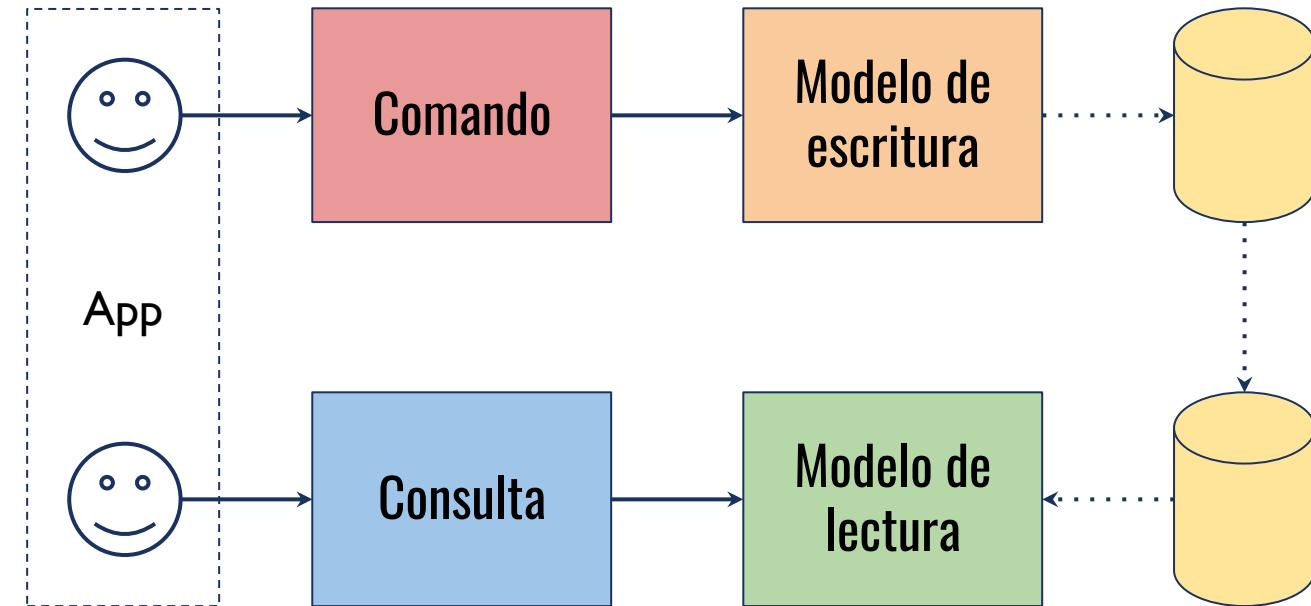


PATRONES ARQUITECTURALES: MICROSERVICIOS / MICROSERVICES

Andrés Armando Sánchez Martín

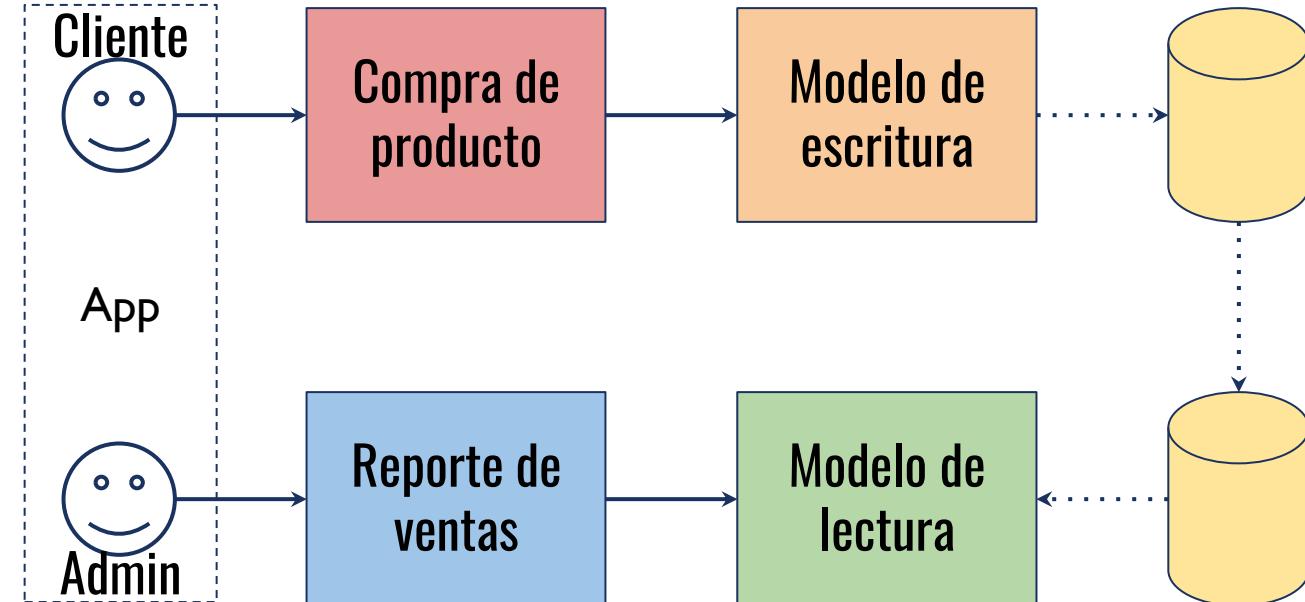


PATRONES ARQUITECTURALES: SEPARACIÓN DE RESPONSABILIDADES ENTRE CONSULTAS Y COMANDOS / CQRS



PATRONES ARQUITECTURALES: SEPARACIÓN DE RESPONSABILIDADES ENTRE CONSULTAS Y COMANDOS / CQRS

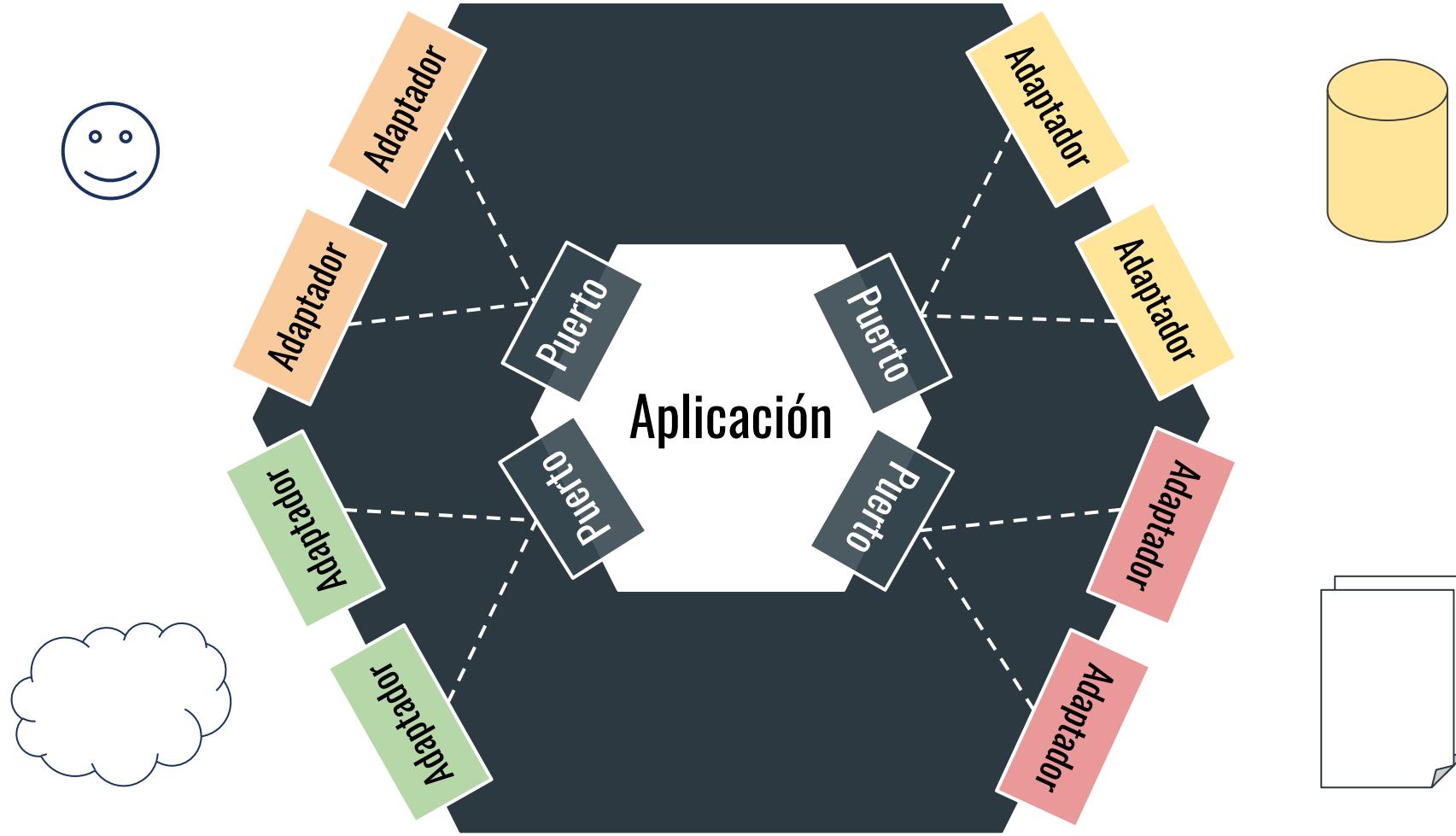
Andrés Armando Sánchez Martín



PATRONES ARQUITECTURALES: HEXAGONAL - PUERTOS Y ADAPTADORES

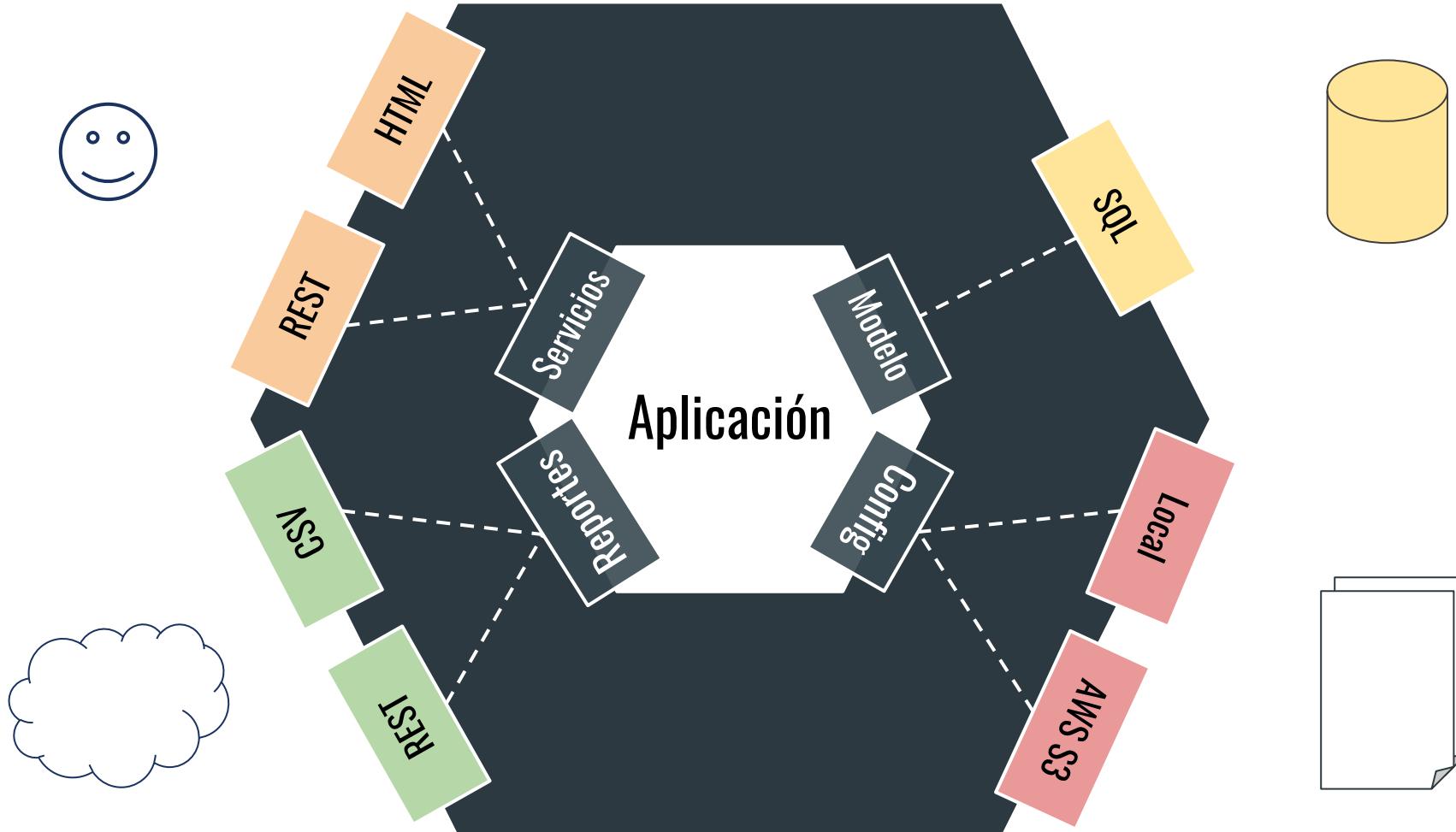
Andrés Armando Sánchez Martín

15

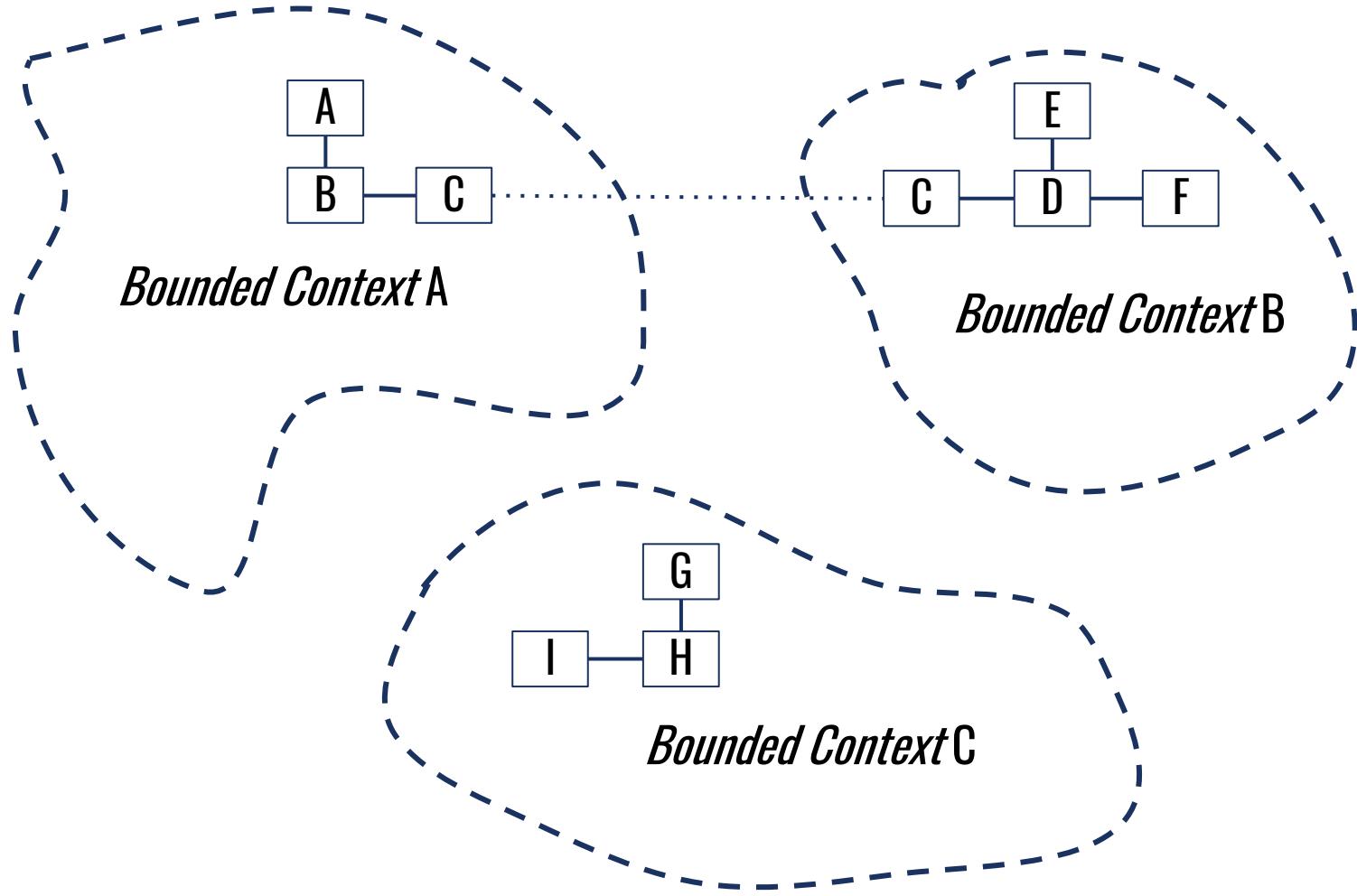


PATRONES ARQUITECTURALES: HEXAGONAL - PUERTOS Y ADAPTADORES

Andrés Armando Sánchez Martín

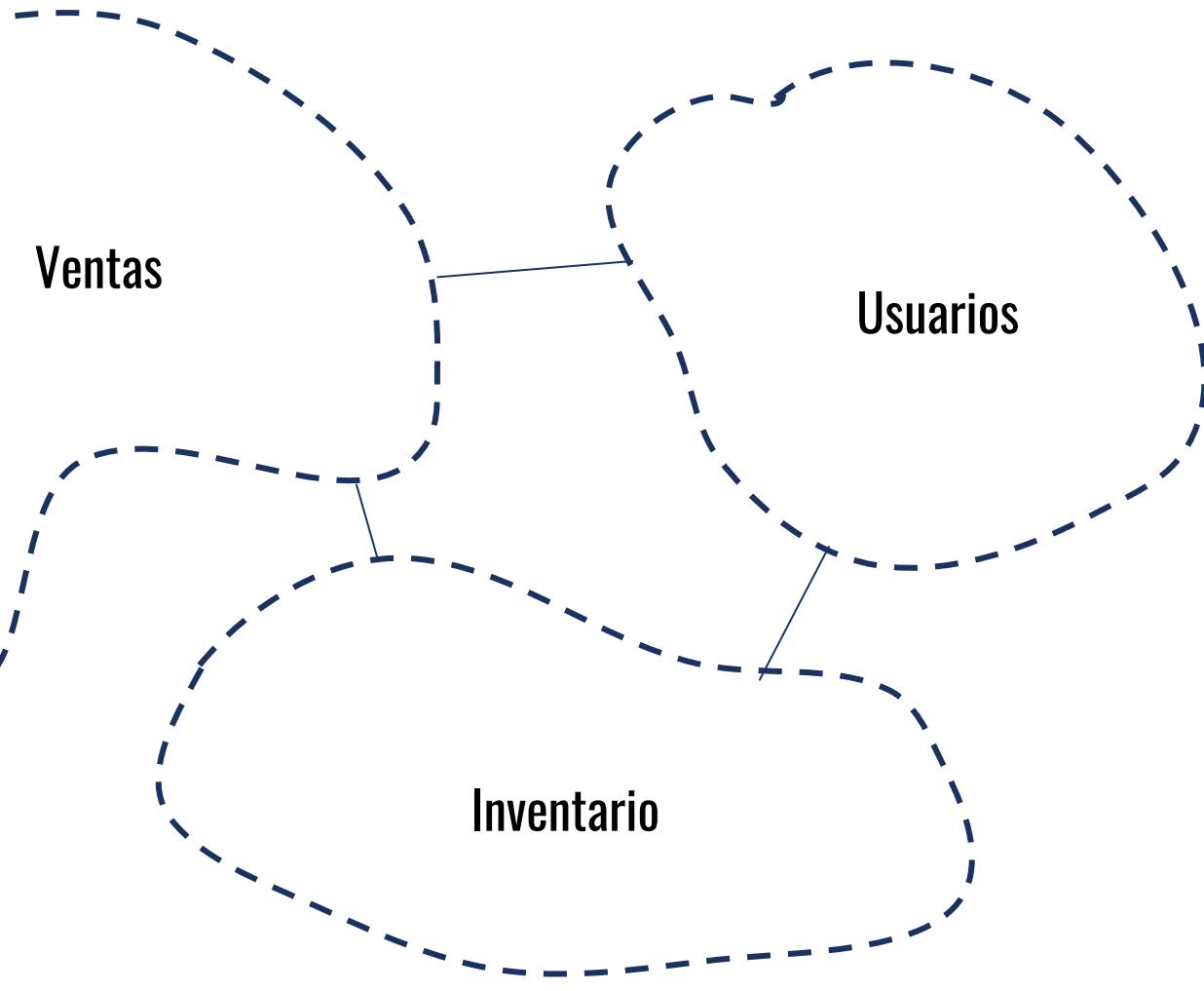


PATRONES ARQUITECTURALES: DISEÑO ORIENTADO AL DOMINIO - DOMAIN-DRIVEN DESIGN



PATRONES ARQUITECTURALES: DISEÑO ORIENTADO AL DOMINIO - DOMAIN-DRIVEN DESIGN

Andrés Armando Sánchez Martín



ESTILOS ARQUITECTURALES

Andrés Armando Sánchez Martín

Llamado y retorno

Programa principal y subrutinas

Orientado a objetos

Cliente-Servidor, Multi-nivel

Flujo de datos

Lote secuencial

Tubos y filtros

Centrado en datos

Pizarrón

Centrado en base de datos

Sistema experto - Basado en reglas

Componentes independientes

Invocación implícita

- Basado en eventos
- Publicar/Suscribir
- Orientado a Servicios 2.0,

Invocación explícita

- Orientado a Servicios 1.0



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition. 2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer. 2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin. Wiley. 2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley. 2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

Andrés Armando Sánchez Martín

- ¿Qué otros estilos de arquitectura ?
- ¿Cómo se llega a un diseño de arquitectura?





(1306)

ARQUITECTURA DE SOFTWARE

5.1 Estilos y Patrones Arquitecturales 3

Agenda

Andrés Armando Sánchez Martín

- Estilos Arquitecturales



ESTILOS ARQUITECTURALES

Andrés Armando Sánchez Martín

Llamado y retorno

Programa principal y subrutinas

Orientado a objetos

Cliente-Servidor, Multi-nivel

Flujo de datos

Lote secuencial

Tubos y filtros

Centrado en datos

Pizarrón

Centrado en base de datos

Sistema experto - Basado en reglas

Componentes independientes

Invocación implícita

- Basado en eventos
- Publicar/Suscribir
- Orientado a Servicios 2.0,

Invocación explícita

- Orientado a Servicios 1.0

ESTILOS ARQUITECTURALES: LLAMADO Y RETORNO

Andrés Armando Sánchez Martín

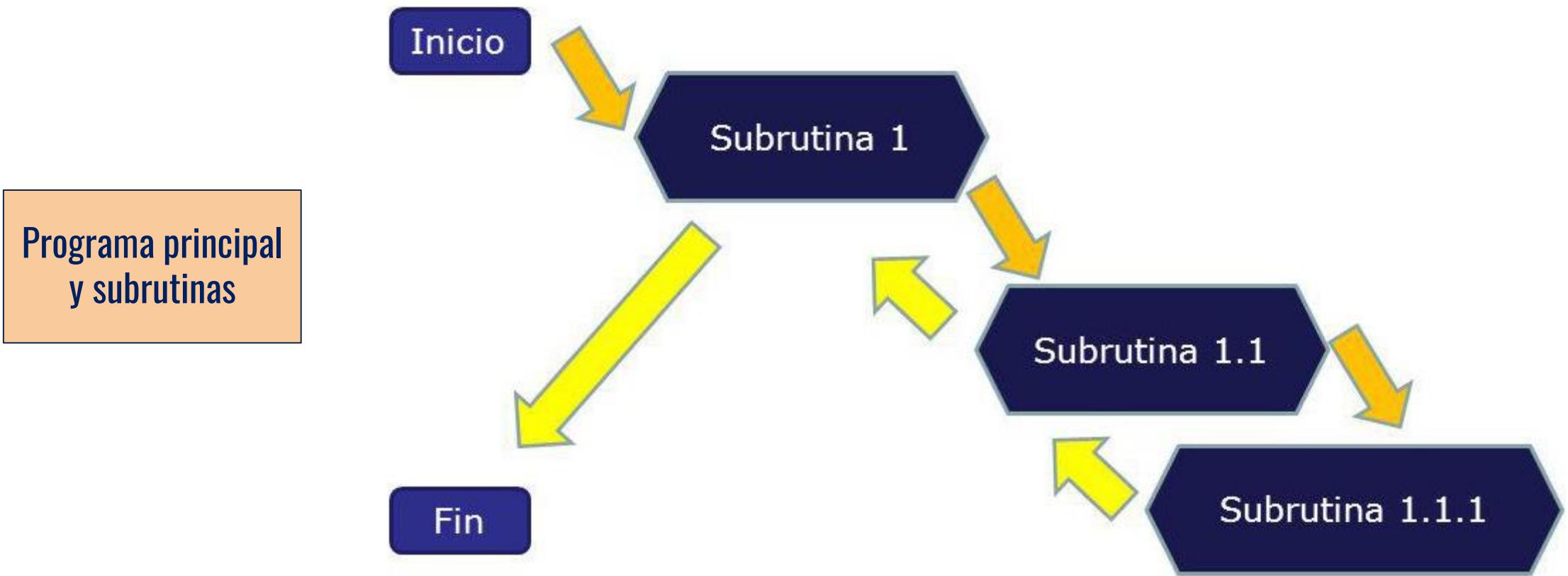
Programa principal
y subrutinas

Orientado a objetos

Multi-nivel

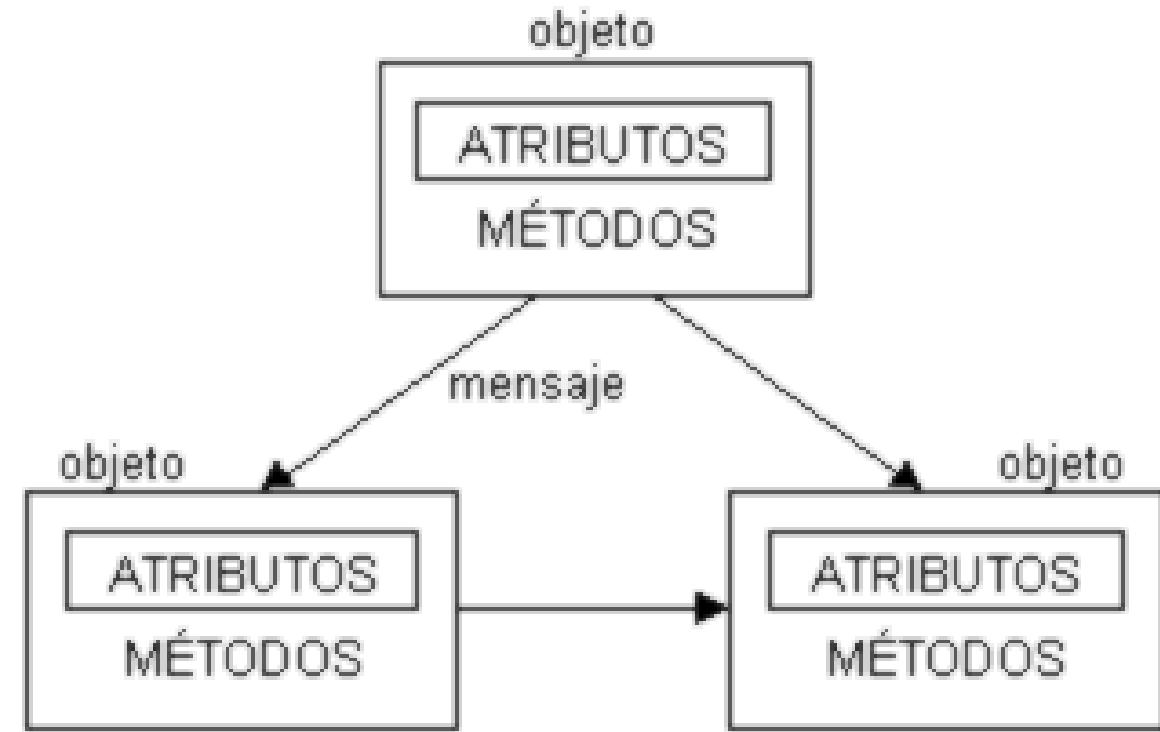
Cliente-Servidor

ESTILOS ARQUITECTURALES: LLAMADO Y RETORNO



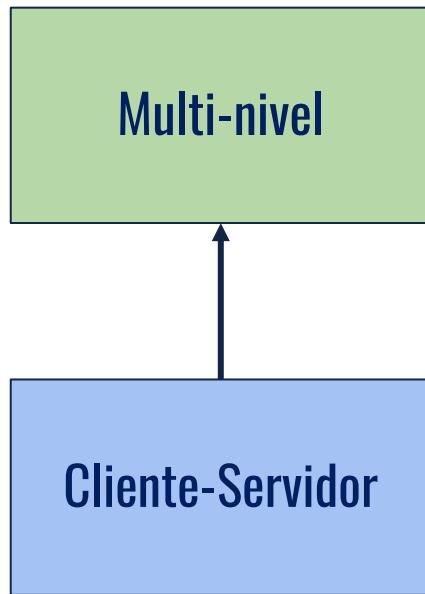
ESTILOS ARQUITECTURALES: LLAMADO Y RETORNO

Orientado a objetos



ESTILOS ARQUITECTURALES: LLAMADO Y RETORNO

Andrés Armando Sánchez Martín





Pontificia Universidad
JAVERIANA
Bogotá

Andrés Armando Sánchez Martín

ESTILOS ARQUITECTURALES: FLUJO DE DATOS

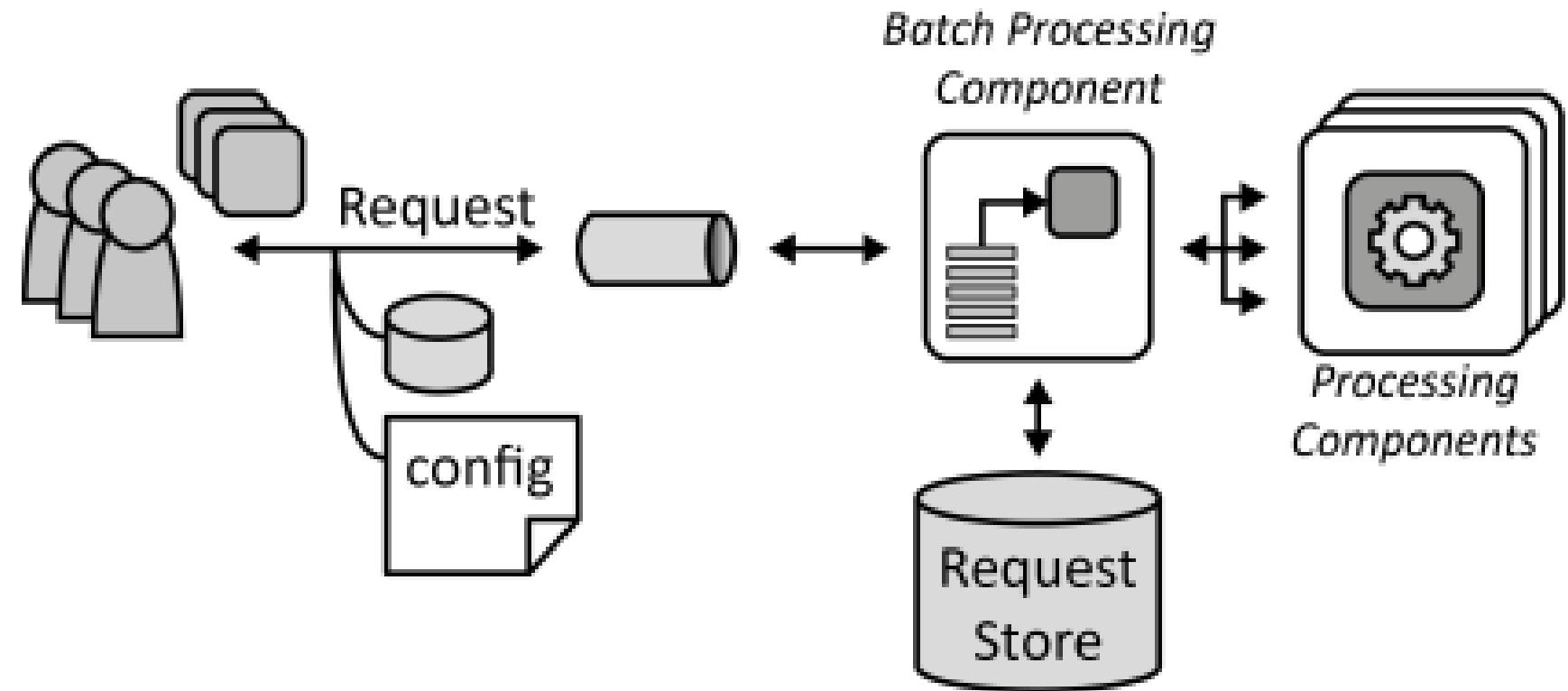
Lote secuencial

Tubos y filtros

ESTILOS ARQUITECTURALES: FLUJO DE DATOS

Andrés Armando Sánchez Martín

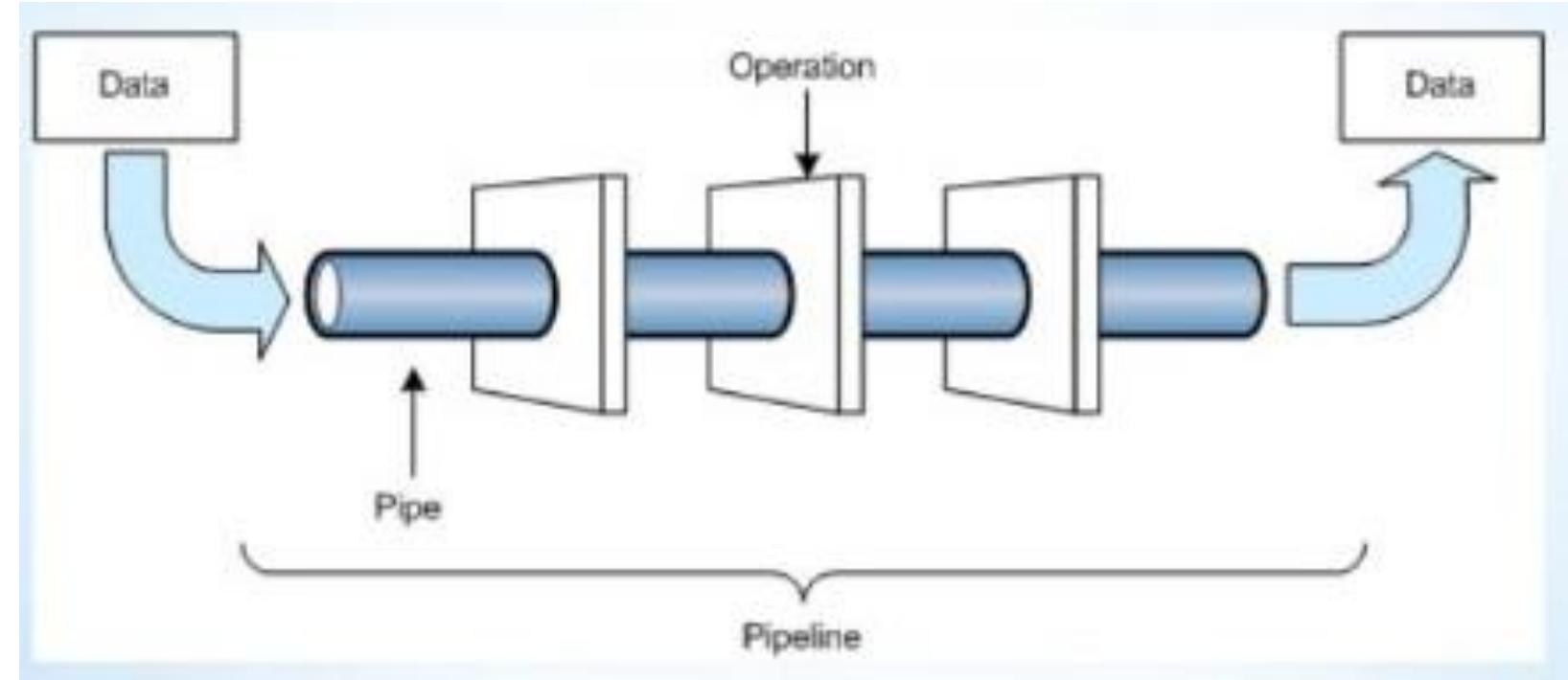
Lote secuencial



ESTILOS ARQUITECTURALES: FLUJO DE DATOS

Andrés Armando Sánchez Martín

Tubos y filtros





Pontificia Universidad
JAVERIANA
Bogotá

Andrés Armando Sánchez Martín

ESTILOS ARQUITECTURALES: CENTRADAS EN DATOS

Pizarrón

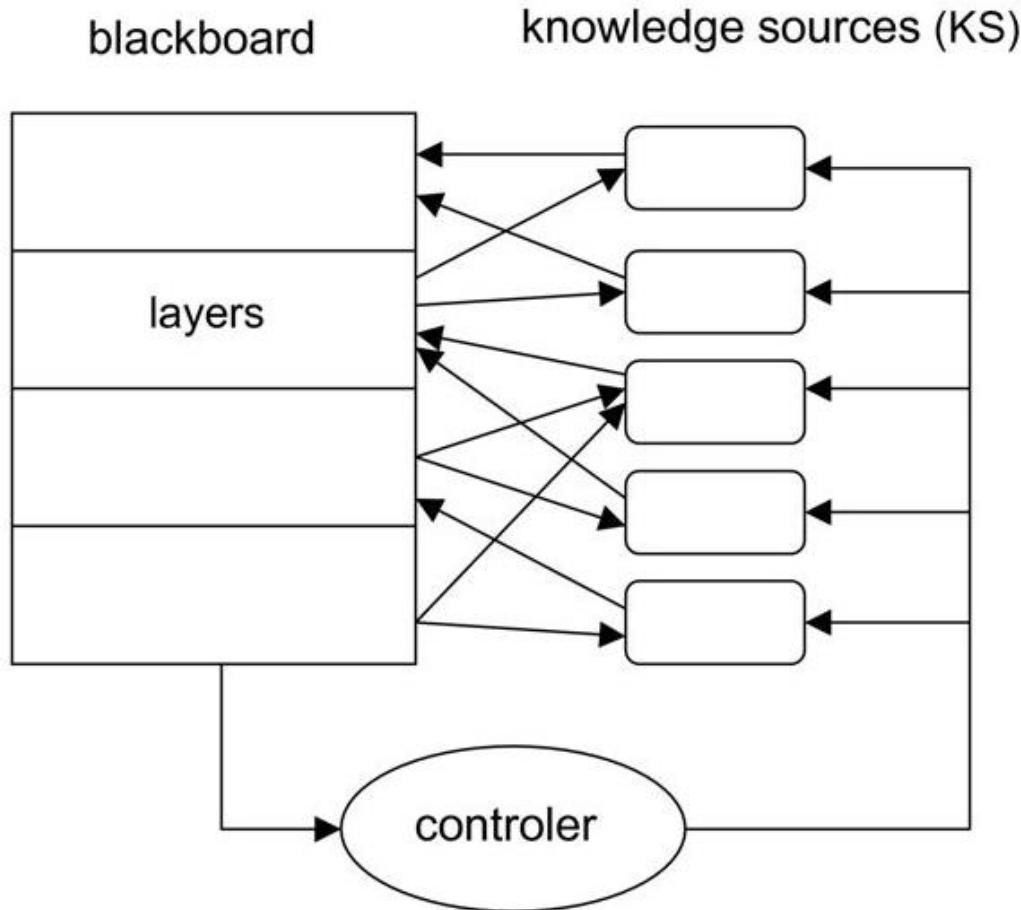
Centrado en
base de datos

Sistema experto -
Basado en reglas

ESTILOS ARQUITECTURALES: CENTRADAS EN DATOS

Andrés Armando Sánchez Martín

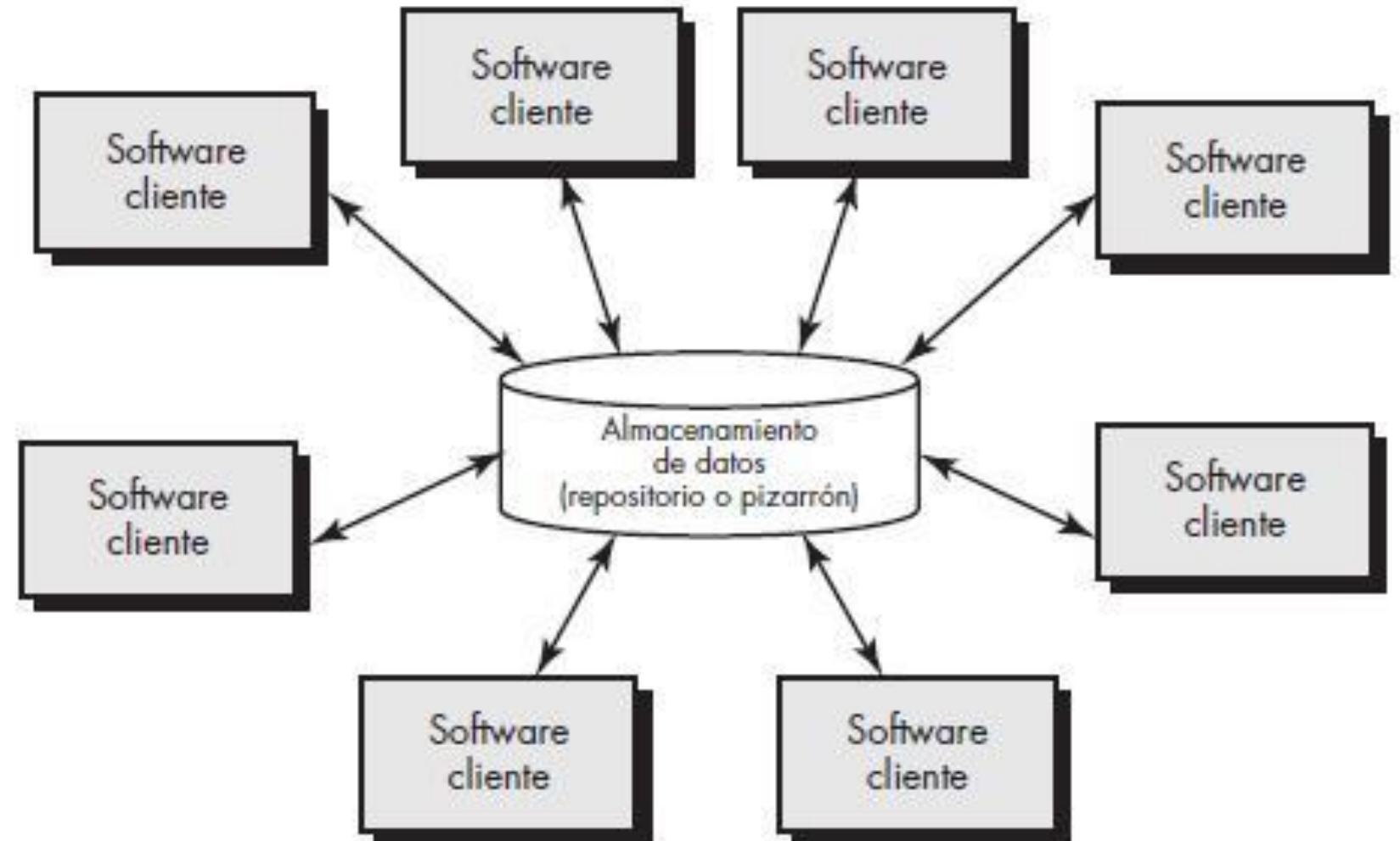
Pizarrón



ESTILOS ARQUITECTURALES: CENTRADAS EN DATOS

Andrés Armando Sánchez Martín

Centrado en
base de datos



ESTILOS ARQUITECTURALES: CENTRADAS EN DATOS

Andrés Armando Sánchez Martín

Sistema experto -
Basado en reglas

↓
Experto

**Módulo de
Adquisición del
Conocimiento**

Motor de Inferencia

Base
de
Datos
(Hechos)

Base de
Conocimientos
(Reglas)

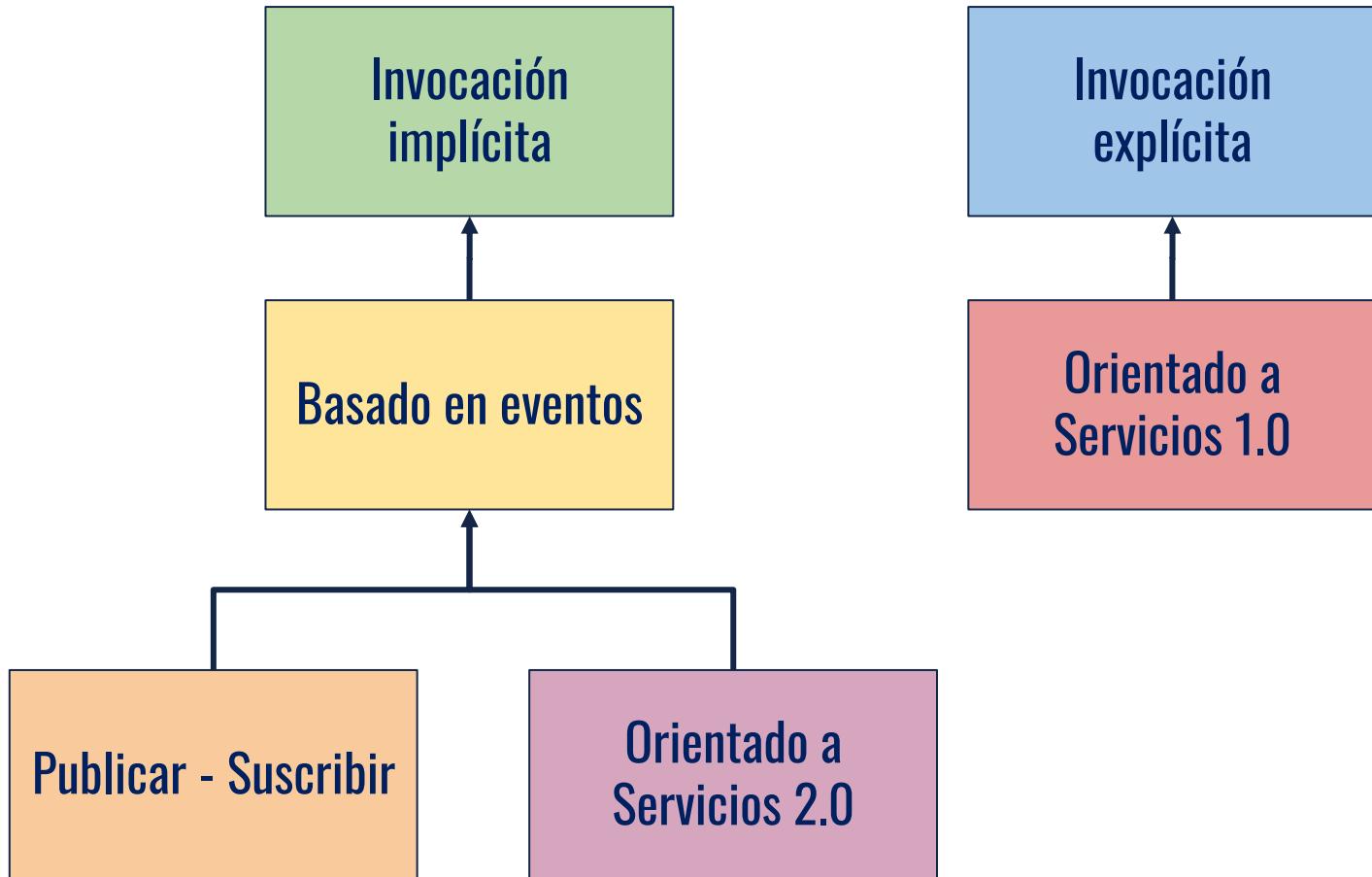
Módulo de
Explicación

**Interface
de
Usuario**



Usuario

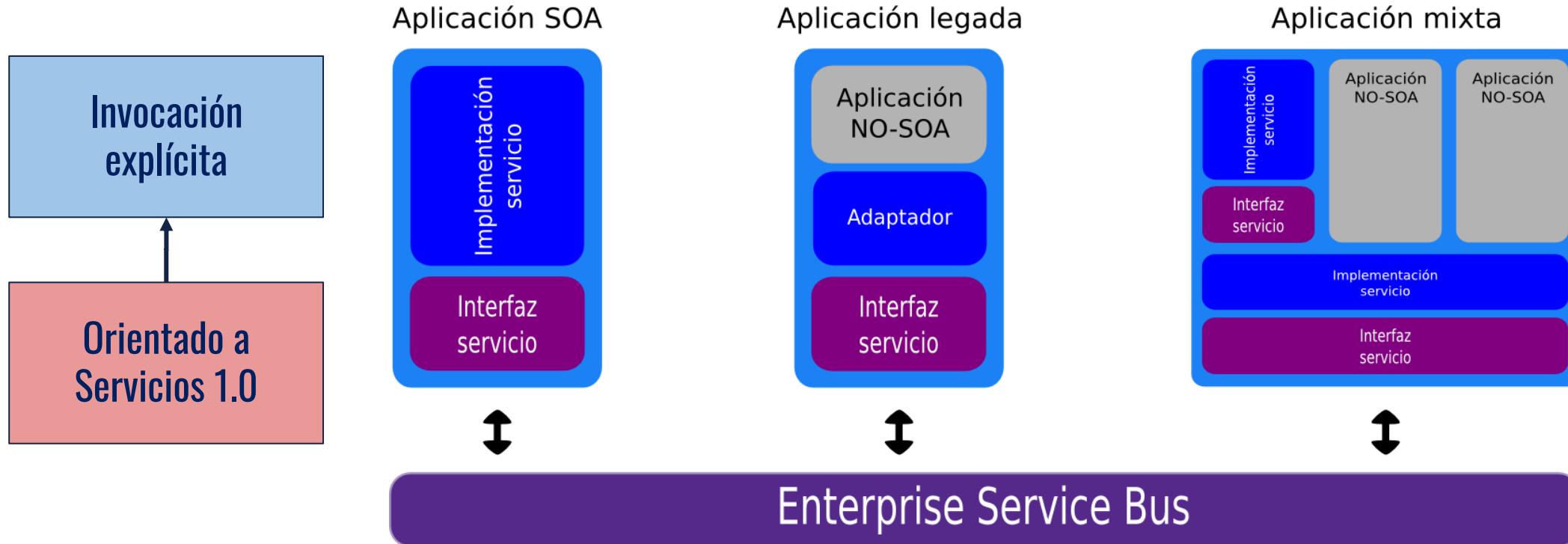
ESTILOS ARQUITECTURALES: COMPONENTES INDEPENDIENTES



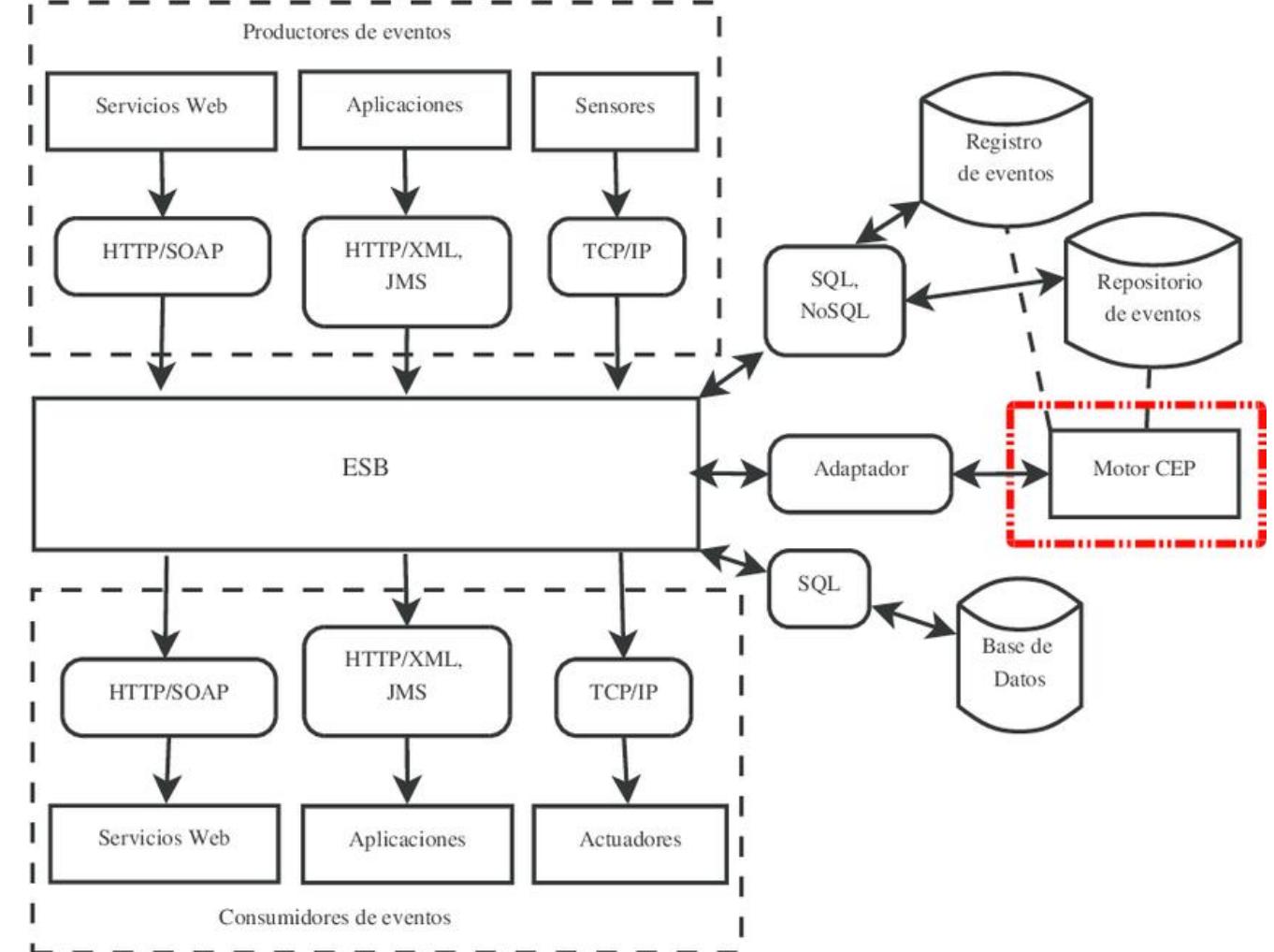
ESTILOS ARQUITECTURALES: COMPONENTES INDEPENDIENTES

Tipos de aplicaciones SOA

Andrés Armando Sánchez Martín



ESTILOS ARQUITECTURALES: COMPONENTES INDEPENDIENTES





BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition. 2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer. 2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin. Wiley. 2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley. 2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Cómo se llega a un diseño de arquitectura?
- ¿Cuáles son las fases de diseño de una Arquitectura?





(1306)

ARQUITECTURA DE SOFTWARE

5.2 Estilos y Patrones Arquitecturales 4

Agenda

Andrés Armando Sánchez Martín

- Como diseñar una Arquitectura.
- Combinando estilos y patrones.
- Casos de Estudio



DISEÑO DE UNA ARQUITECTURA

- La disciplina evoluciona
 - Arquitecto debe conocer:
 - Avances en técnicas de construcción
 - Estilos y patrones
 - Mejor herramienta = experiencia (no silver bullet)
 - Experiencia propia
 - Experiencia de la comunidad
 - Construir vs reutilizar
 - En algunos dominios, reutilizar arquitecturas existentes puede ser más eficiente
 - Arquitecturas de referencia
 - Componentes desarrollados externamente



DISEÑO DE UNA ARQUITECTURA

- Pararse en hombros de gigantes
- Aprovechar el conocimiento existente para nuestra solución.

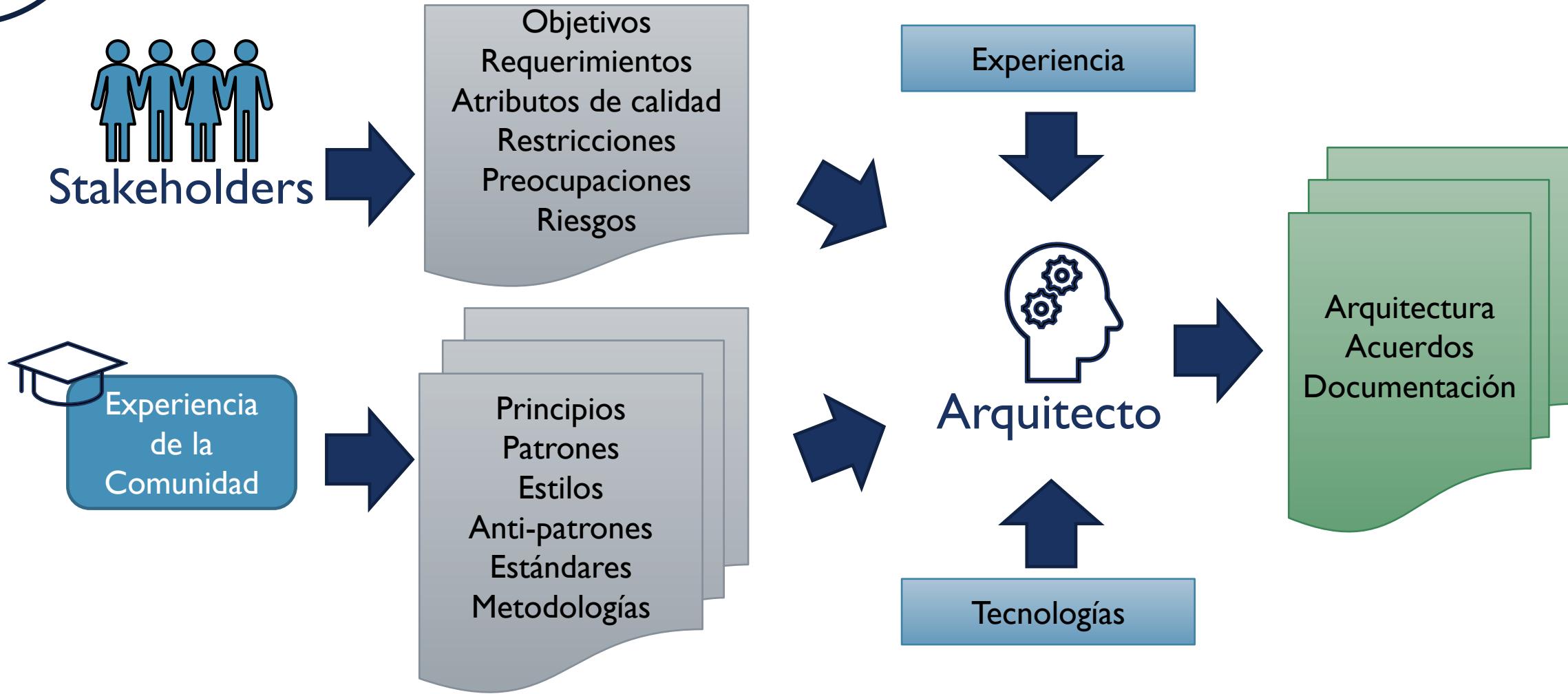
Productos
“de la estantería”

Frameworks

Arquitecturas
específicas del
dominio

Patrones de
arquitectura

PROCESO DE ARQUITECTURA: ARQUITECTO



DISEÑO DE UNA ARQUITECTURA

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

Arquitecturas de software de dominios específicos

- Combinación de:
 - Arquitectura de referencia para un dominio
 - Librería de componentes para dicha arquitectura
 - Método para elegir y configurar componentes para trabajar dentro de una instancia de dicha arquitectura de referencia
- Especializados para un dominio concreto
- Ejemplos:
 - ADAGE, MetaH

DISEÑO DE UNA ARQUITECTURA

Componentes desarrollados externamente

- Pilas tecnológicas o familias (Stack)
 - 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

Alcanzando arquitectura del software

- Metodologías
 - ADD
 - Basado en riesgos
 - Basado en aspectos
- Toma de decisiones
- Incidencias arquitectónicas
 - Riesgos
 - Desconocidos
 - Problemas
 - Restricciones
 - Deuda técnica
 - Diferencias en comprensión
 - Erosión / Ir a la deriva
- Evaluación de arquitecturas

DISEÑO DE UNA ARQUITECTURA: 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 se pueden usar
RFC (Request for comments)

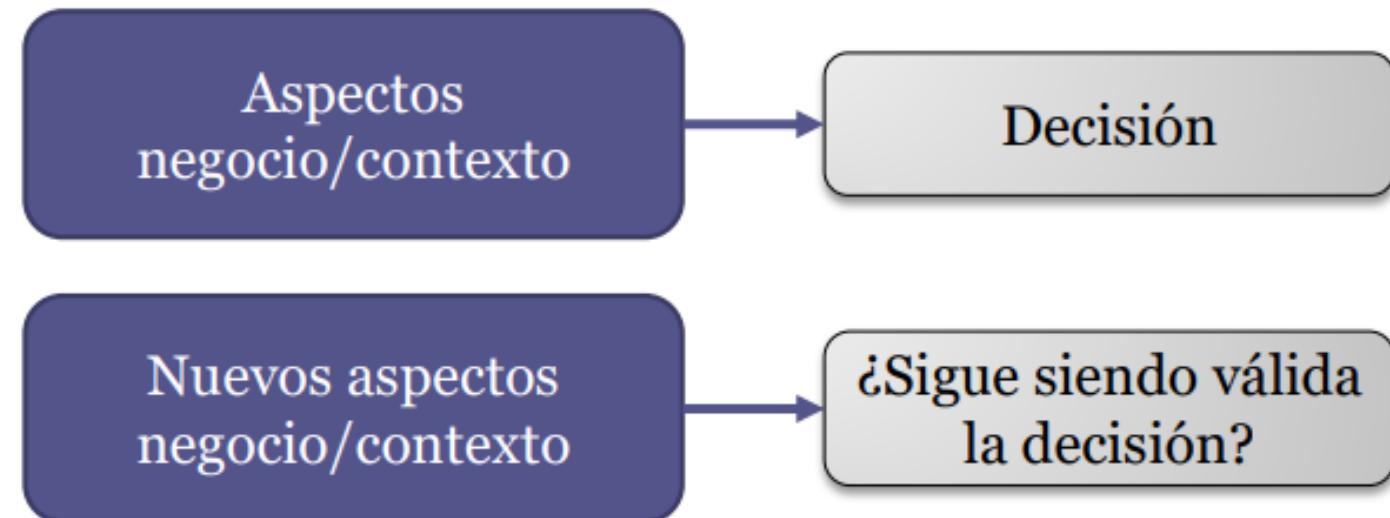
DISEÑO DE UNA ARQUITECTURA

■ 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

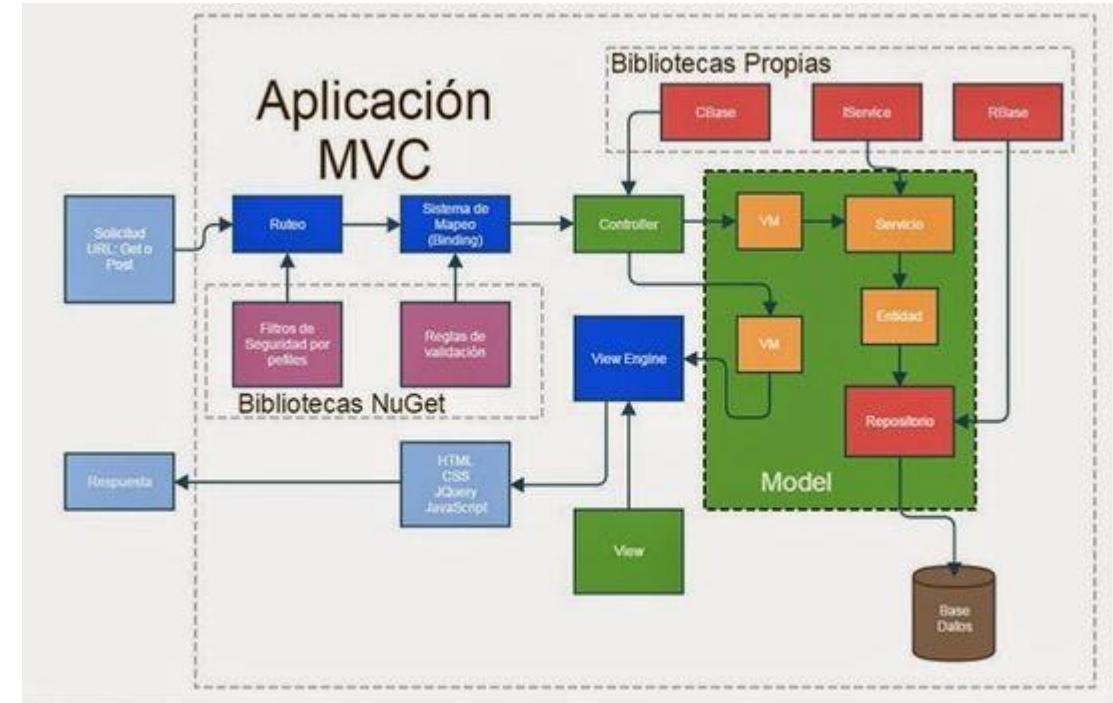
■ Evolución del contexto

- Ocurre cuando aspectos claves del contexto cambian después de haber tomado una decisión de diseño
 - Es necesario revisar continuamente los requisitos
 - Arquitecturas evolutivas



COMBINANDO ESTILOS Y PATRONES.

- Una solución debe contar con un estilo de arquitectura
 - Este puede ser un estilo puro
 - Pero más comúnmente es una mezcla de estilos
- Una solución puede incluir múltiples patrones
 - Tantos como problemas comunes se identifiquen
- No existe una solución estándar para la solución
 - Si bien los contextos pueden ser similares
 - El dominio del problema siempre es único



COMBINANDO ESTILOS Y PATRONES: PROCESO DE DISEÑO

Diseño arquitectónico

Diseño de alto nivel

Diseño detallado

Plataforma de información

Especificación de requerimientos

Descripción de datos

Diseño arquitectónico

Diseño de interfaz

Diseño de componentes

Diseño de base de datos

Entradas de diseño

Actividades de diseño

Salidas de diseño

Arquitectura del sistema

Especificación de base de datos

Especificación de interfaz

Especificación de componentes

COMBINANDO ESTILOS Y PATRONES: PROCESO DE DISEÑO

Diseño de Arquitectura

Restricciones
Riesgos
Requerimientos
Atributos de Calidad

Diseño Lógico

Estilo Arquitectural
UML
Modelo de datos

Diseño de Detalle

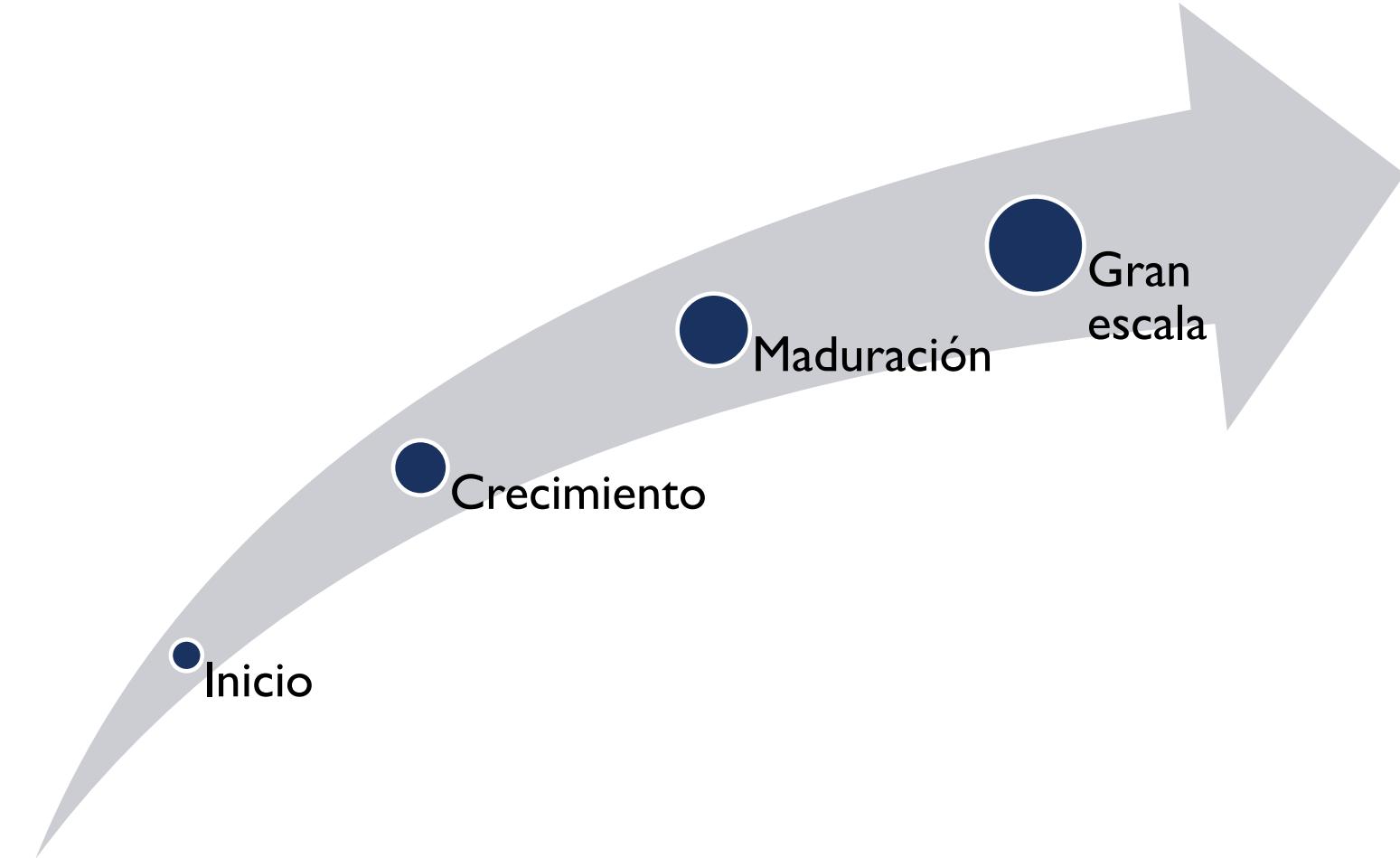
4 Vistas mas I
Modelo C4



Pontificia Universidad
JAVERIANA
Bogotá

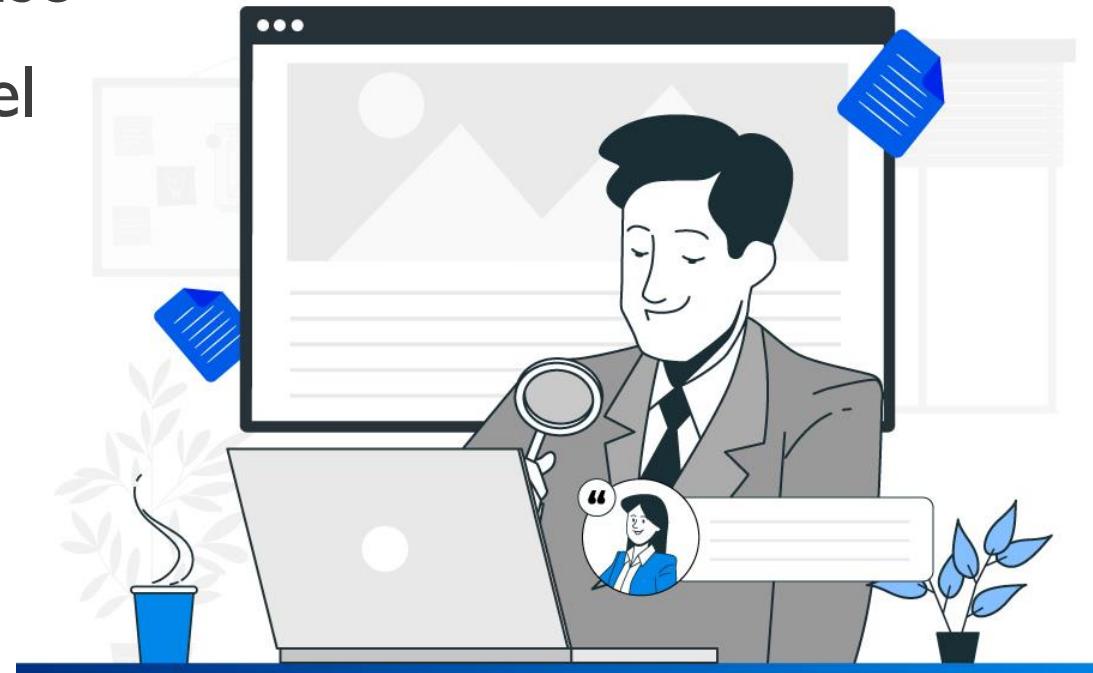
ETAPAS DE UNA SOLUCIÓN

Andrés Armando Sánchez Martín



CASO DE ESTUDIO

- Desarrollar diagrama de casos de uso
- Desarrollar el diagrama de alto nivel
- Definir matriz de acuerdos
- Identificar:
 - Distribución geográfica
 - Interesados
 - Expectativas de usuarios
 - Estilos involucrados
 - Patrones involucrados



CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: ALMACÉN Y FACTURA

- Diseñar una herramienta de gestión comercial para Windows. Factura en pesos, USD o Euro. Permite trabajar con lector de códigos de barras (EAN).
- Su arquitectura de 32 bits, mejora su rendimiento. Podrá trabajar de un modo más fácil y rápido, haciendo su empresa más operativa.
- Permite gestionar varios almacenes, un número indefinido de artículos a los cuales se les puede añadir en su ficha una foto del mismo, posibilidad de trabajar con proveedores, agentes, zonas, perfecto control de los vendedores, comisiones, descuentos, y depósitos. Y todo con un sencillo diseño.
- Incorpora mejoras en el tratamiento de inventarios, tarifas, cambios automáticos de precios, listados, informes, varios tipos de facturación, retenciones de I.R.P.F. remesa de recibos en soporte magnético, gestión de cobros, etc.
- El diseño de documentos es configurable por el usuario, incorporación de modelos estándar de documentos, restricción de accesos a la aplicación configurables por el administrador, enlace opcional con la contabilidad CONTAGEWIN, terminal punto de venta TPVWIN compartiendo la misma información, represente todo tipo de estadísticas de clientes, proveedores, agentes, zonas, etc

CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: ALMACÉN Y FACTURA

- Características:
 - Multi-almacén, amplio fichero de artículos, clientes, proveedores, comisiones, descuentos, depósitos.
 - Su diseño permite el enlace desde presupuestos/pedidos, pedidos/albaranes, albaranes/facturas, facturas/recibos.
 - Facturación en pesos, USD, Euro
 - Incorpora mejoras en tratamiento de inventarios, tarifas, cambios automáticos de precios, listados, informes, varios tipos de facturación, retenciones de IR.P.F, remesa de recibos en soporte magnéticos.
 - Gestión de cobros, envío de documentos vía fax desde la aplicación.



CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: DEIMOS

DEIMOS Space tiene como objetivo liderar el campo del desarrollo de software empotrado en tiempo real, que cubre el ciclo de vida completo de desarrollo y producción desde la fase de análisis de requisitos a través del diseño detallado y de arquitectura, codificación, testeo, integración, validación y aceptación. Estos desarrollos de software usan típicamente HOORA/UML para el análisis y modelado de requisitos, HRT-HOOD y Ada/C/C++ para el diseño e implementación.

El alcance de los desarrollos incluye software de aplicación de alto nivel así como servicios básicos de software empotrado para el tratamiento de telemetría y tele-comandos, programación de tareas, detección de fallos, aislamiento y recuperación y controladores de dispositivos.

Además de desarrollos para misiones específicas, DEIMOS Space realiza y participa en estudios tecnológicos dirigidos a mejorar el estado del arte del software empotrado en tiempo real.

- I. Software de control de vuelo: DEIMOS Space tiene capacidad para desarrollar software embarcado de control de vuelo critico en seguridad para: Sistemas de Control de órbita y Altitud (AOCS); Sistemas de Medida y Control de Altitud (ACMS); sistemas de Guiado, Navegación y Control (GNC). Se aplican técnicas de aseguramiento de productos software tales como análisis de Fiabilidad, Disponibilidad, Mantenibilidad y Seguridad (RAMS); Análisis del árbol de Fallos (FTA); Análisis de Criticidad y Efectos de Modo Fallo (FMECA); Análisis de Efectos de Errores Software (SEEA).

CASO DE ESTUDIO ATRIBUTOS DE CALIDAD: DEIMOS

- 2. Software de tratamiento de Datos a Bordo: DEIMOS Space puede desarrollar software para subsistemas de Gestión de Datos a bordo (OBDH) que utiliza memoria de almacenamiento masivo, recogida y envío de telemetría, recepción y distribución de tele-comandos, gestión del interface de la plataforma de vuelo, y tareas de mantenimiento autónomo del sistema.
- 3. Software de control de Instrumentos: DEIMOS Space tiene capacidad para desarrollar software empotrado de tiempo real para ordenadores encargados de controlar instrumentos científicos a bordo de satélites y sondas espaciales. Típicamente el software realiza funciones específicas de comando y control de instrumentos, procesamiento de datos científicos y mantenimiento, control y monitorización del interface del instrumento y gestión de modos.
- 4. Validación de software de Segmento Espacio: DEIMOS Space puede realizar validación de software usando técnicas en línea con el estado del arte y entornos de testeo, e incluyendo métodos de análisis estático y dinámico.
- 5. Simulación de Sistemas Embarcados: DEIMOS Space complementa sus actividades en el rea de software embarcado con experiencia en simulación como medio para apoyar el desarrollo, validación y análisis de productos software. Para mas información acerca de las actividades de DEIMOS Space en el campo de Sistemas en Tiempo Real, por favor contacte con D. Mike Rennie.



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



Pontificia Universidad
JAVERIANA
Bogotá

¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Cuál es el proceso de arquitectura de software?
- ¿Qué es un framework de arquitectura de software?
- ¿Cuáles son los frameworks y herramientas de la arquitectura de software?
- ¿Qué tiene de nuevo la especificación de UML 2.5?

Taller:

- Entregar documento con la identificación de : distribución geográfica, interesados, expectativas de usuarios, estilos involucrados, patrones involucrados.
- Además de esto, para cada caso de estudio hacer: Diagramas de casos de uso, diagrama de alto nivel de la arquitectura y definir matriz de acuerdos
- Entrega clase 6.2





(1306)

ARQUITECTURA DE SOFTWARE

6.1 Proceso de Arquitectura de Software I

Agenda

- Frameworks y Herramientas de Arquitectura de software:
 - IEEE 1471 -> ISO / IEC / IEEE 42010
 - Arquitectura Empresarial: TOGAF y Zachman
 - Gobierno de TI: ITIL y COBIT
 - Software Engineering Institute (SEI): CMMI, TSP y PSP
 - Metodologías Agiles: Scrum y XP
 - UML - Lenguaje Unificado de Modelado
 - DDD - Domain-Driven Design
 - Basado en vistas: Kruchten's 4+1 View Model y C4 Model
 - Estructura del Documento de Diseño de Software (SDD)



PROCESO DE ARQUITECTURA DE SOFTWARE

Identificar y especificar los requerimientos guía. Éstos incluyen a los atributos de calidad, los requerimientos funcionales primarios, los riegos y las restricciones del sistema. La arquitectura se puede diseñar alrededor de estos.

Diseñar la arquitectura e, idealmente, implementar una “arquitectura ejecutable” que permita materializar el diseño de la arquitectura.

Documentar los aspectos fundamentales del diseño, teniendo especial cuidado en capturar las decisiones (acuerdos) de diseño, para comunicarlas al equipo y que sirvan de guía.

Realizar una evaluación poco después de que se ha terminado el diseño y antes de proceder a la construcción del sistema.



PROCESO DE ARQUITECTURA DE SOFTWARE

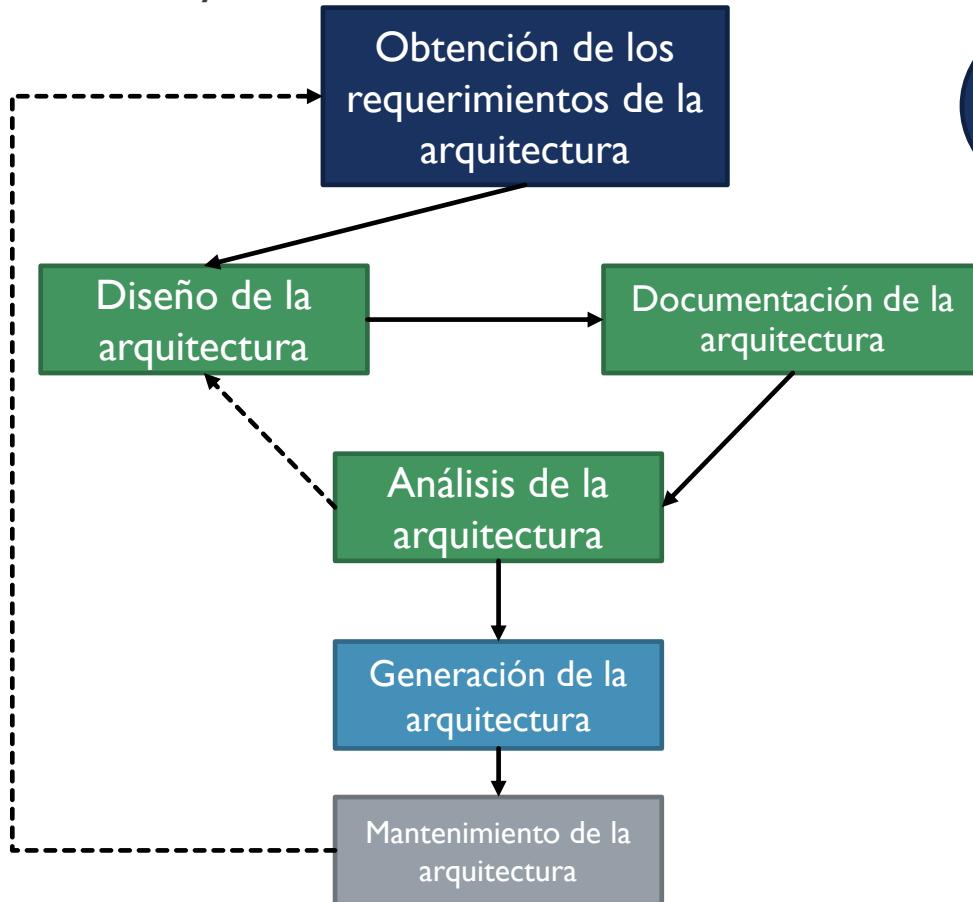
Andrés Armando Sánchez Martín





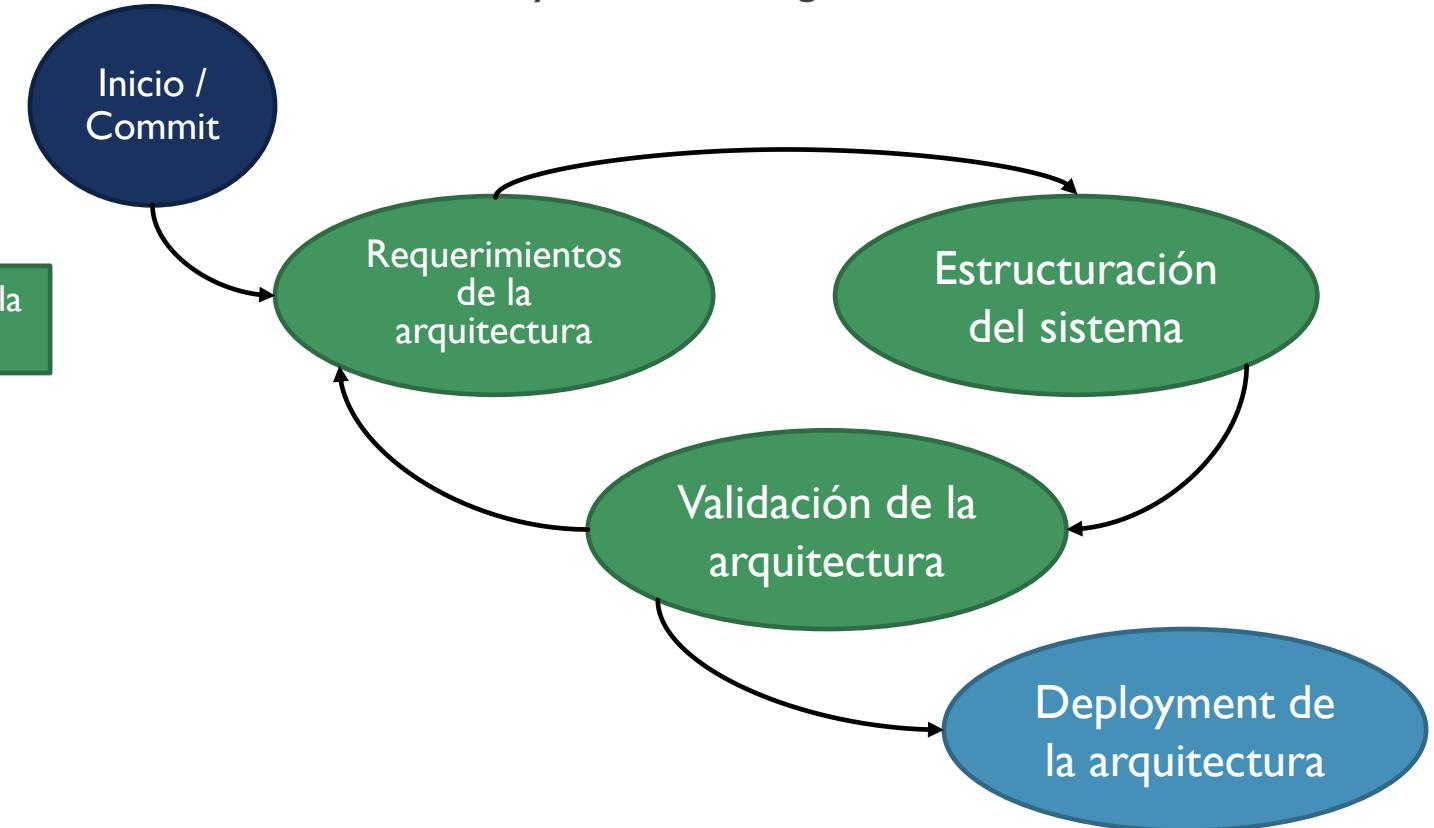
PROCESO DE ARQUITECTURA DE SOFTWARE: MODELOS

■ Bass y Kazman



Andrés Armando Sánchez Martín

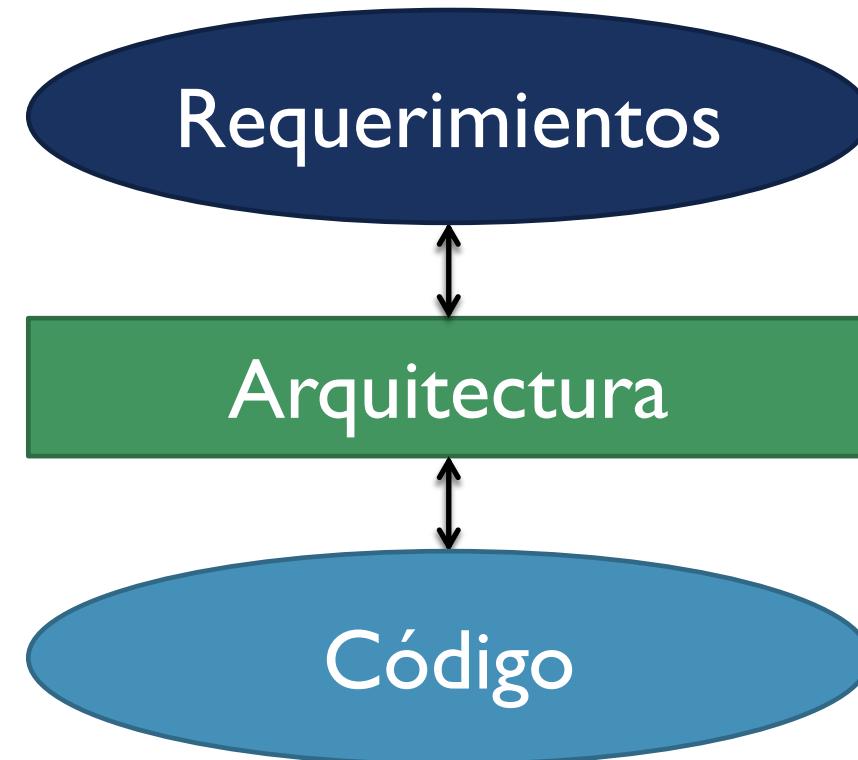
■ Bredemeyer Consulting





Pontificia Universidad
JAVERIANA
Bogotá

PROCESO DE ARQUITECTURA DE SOFTWARE





FRAMEWORKS Y HERRAMIENTAS DE ARQUITECTURA DE SOFTWARE

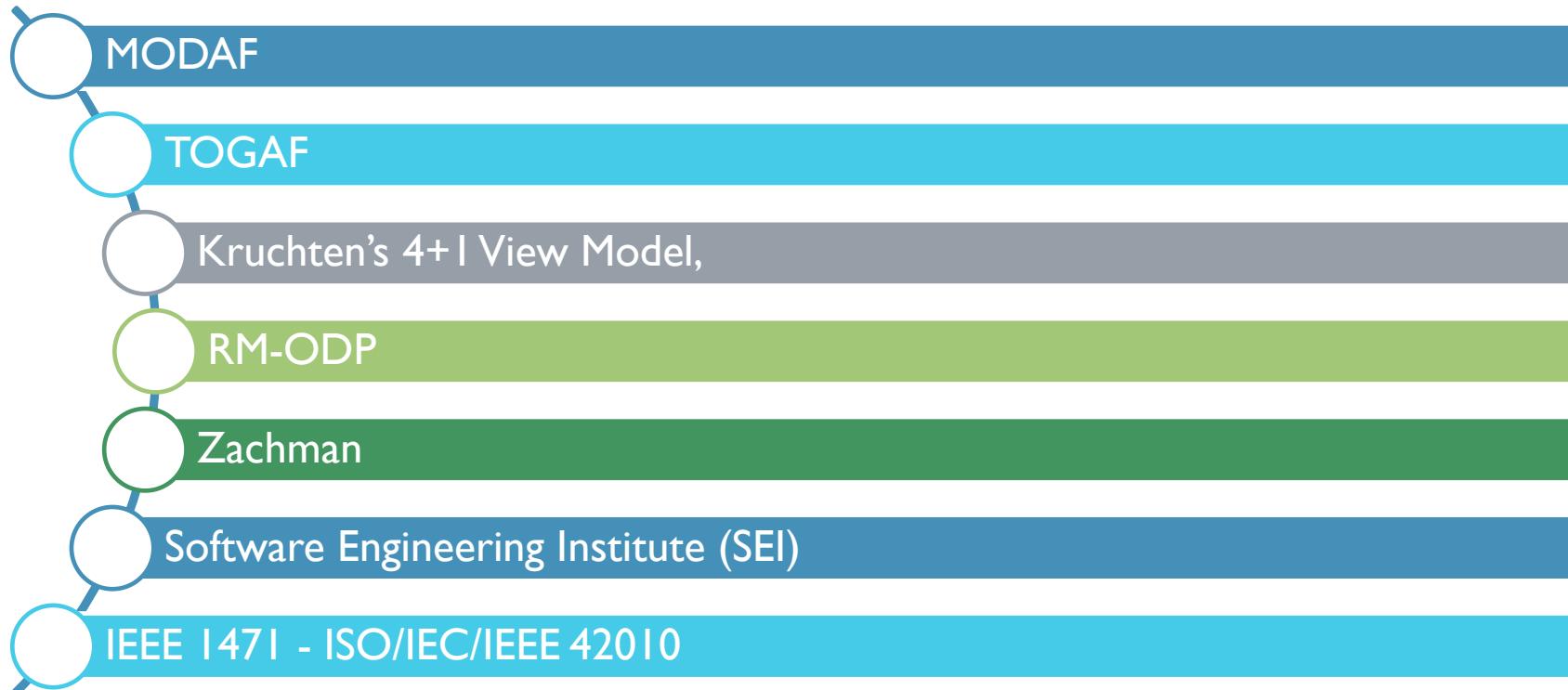
Frameworks

- Procesos
- Actividades
- Flujos
- Artefactos

Herramientas

- Lenguajes
- Metodologías
- Normas
- Estándares
- Aplicaciones CASE

FRAMEWORK DE ARQUITECTURA DE SOFTWARE



Andrés Armando Sánchez Martín

"convenciones, principios y prácticas para la descripción de arquitecturas establecidas dentro de un dominio específico de aplicación y/o comunidad de partes interesadas."

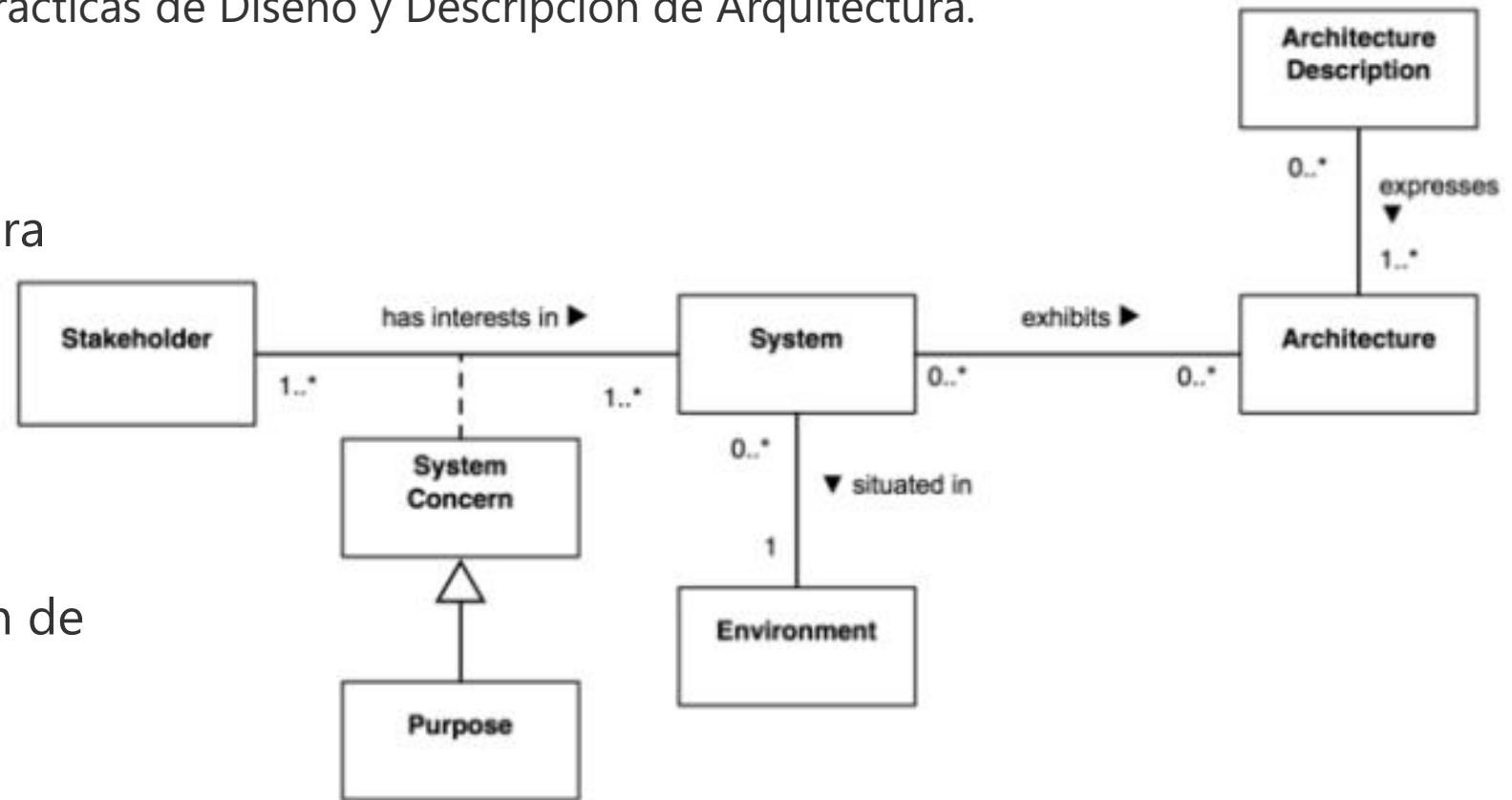
IEEE 1471 : ISO/IEC/IEEE 42010 Conceptual Model of Architecture Description (2011)

IEEE 1471 - ISO/IEC/IEEE 42010

Establece requisitos para describir la arquitectura de sistemas y software a través de una convención, terminología común y mejores prácticas de Diseño y Descripción de Arquitectura.

Define:

1. Descripción de la arquitectura
2. Punto de vista de la arquitectura
3. Marco de trabajo para la arquitectura
4. Lenguaje para la descripción de arquitectura





IEEE 1471 - ISO/IEC/IEEE 42010

Andrés Armando Sánchez Martín

Arquitecturar

Arquitectura

Descripción de la arquitectura (AD)

Lenguaje para la Descripción de Arquitectura (ADL)

Asunto

Marco de trabajo de arquitectura

Punto de vista de la arquitectura

Vista de la arquitectura

IEEE Std
1471:2000

ISO/IEC
42010:2007

ISO/IEC/IEEE
42010:2011

ARQUITECTURA EMPRESARIAL

La arquitectura empresarial es una práctica en la gestión empresarial y en la de tecnologías de la información (TI).

Esta enfocada en mejorar el desempeño de una institución al entenderla en términos integrales desde su perspectiva estratégica, desde las prácticas y procesos organizacionales, y a partir de las TI como habilitadoras de la entidad.



Arquitectura empresarial

- Estructura y comportamientos de un negocio
- Roles y procesos de negocios
- Comprende IT y diseño de la organización

2 técnicas principales

- Basada en modelo
 - Zachman framework
- Basada en iniciativas
 - TOGAF

ARQUITECTURA EMPRESARIAL

Componentes de la Arquitectura Empresarial

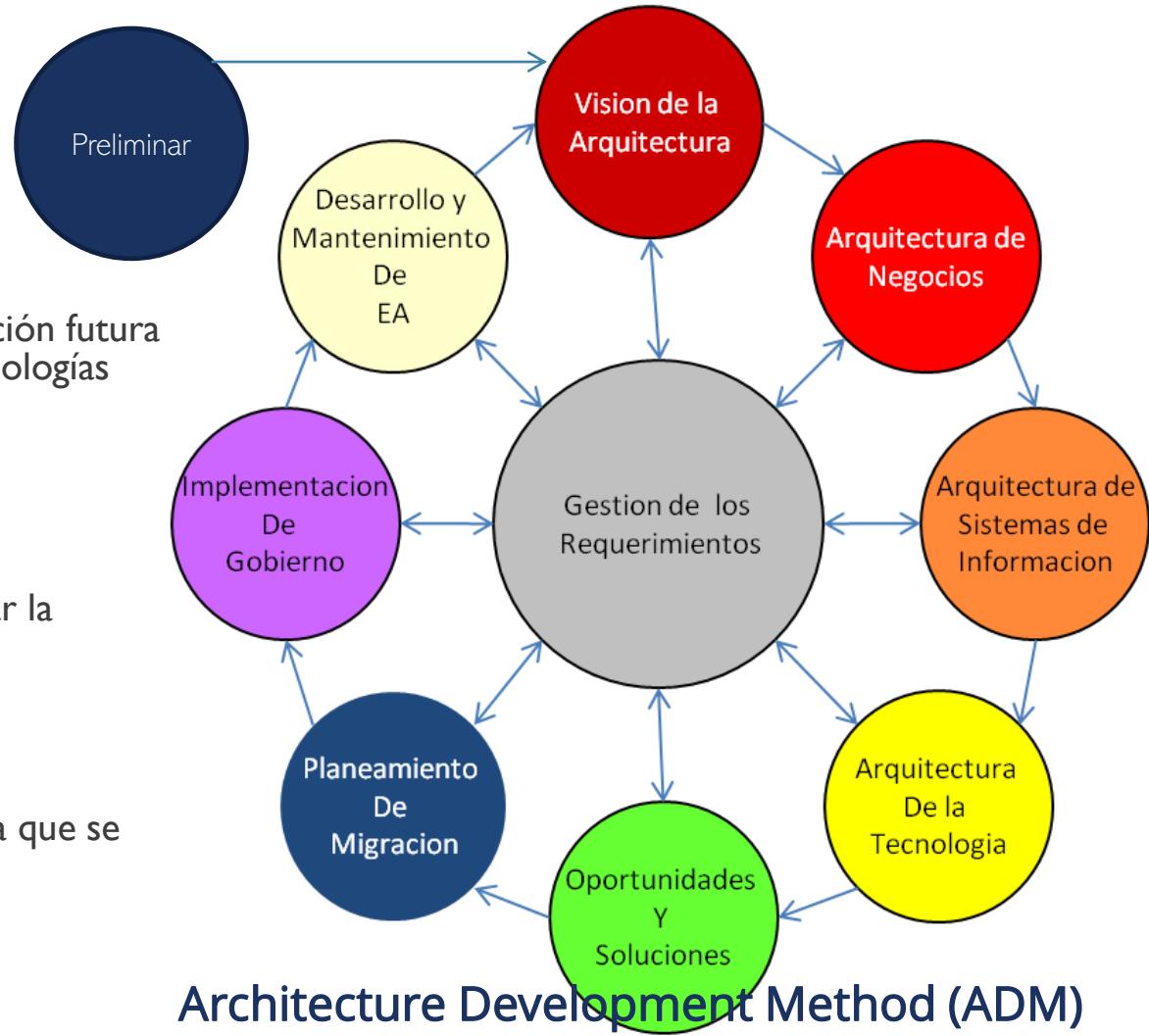


ARQUITECTURA EMPRESARIAL: ZACHMAN FRAMEWORK

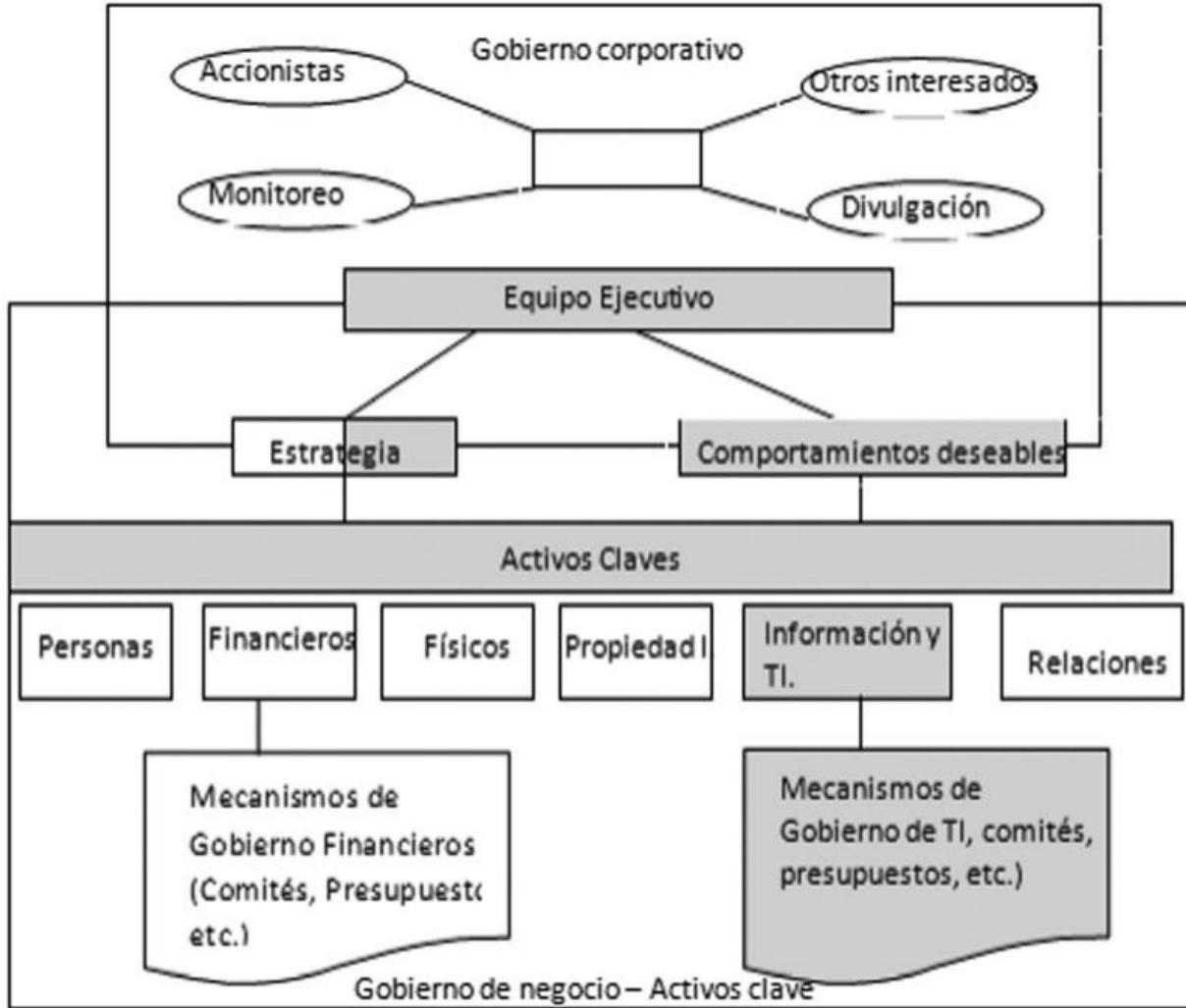
	Datos Qué	Procesos Cómo	Redes Dónde	Personas Quiénes	Tiempo Cuándo	Motivo Por qué	
Ámbito (Entorno)	Lista de intenciones del negocio	Lista Procesos hacia afuera	Lista de ámbitos del Negocio	Lista de Stakeholders	Lista de datos del negocio	Lista de metas del negocio	Gerencia
Modelo Empresa (Conceptual)	Modelo Semántico del negocio	Modelo de Procesos de Negocio	Modelo de interacción	Modelo perfil-rol	Plan maestro	Plan Estratégico de negocio	Dueño
Modelo de Sistemas de Información (Lógico)	Modelo Lógico de Datos	Arquitectura del Sistema de Información	Arquitectura distribuida del sistema de información	Arquitectura de Interfaces Humanas	Estructura de Procesamiento	Modelo de reglas de negocio	Diseñador
Modelo Tecnológico (Físico)	Modelo Físico de Datos	Diseño de Aplicaciones	Arquitectura Tecnológica	Arquitectura de presentación	Estructura de Control	Diseño de reglas de negocio	Constructor
Especificaciones de detalle (producto)	Especificación de Datos	Programas	Arquitectura de Redes	Arquitectura de Seguridad	Definición de tiempos	Especificación de reglas de negocio	Sub contratista
Empresa funcionando	Datos	Procesos	Redes	Organización	Programa	Estrategias	Empresa funcionando

ARQUITECTURA EMPRESARIAL: THE OPEN GROUP ARCHITECTURE FRAMEWORK (TOGAF)

- Contexto de la arquitectura: Identificar la estrategia de la organización y obtener su compromiso y participación
 - Preliminar
 - Visión de la arquitectura
- Entrega de la arquitectura: Entender la situación actual y diseñar la situación futura con relación de los dominios de negocio, sistemas de información y tecnologías
 - Arquitectura de negocios
 - Arquitectura de sistemas de información
 - Arquitectura de la tecnología
- Planificación de la transición: Identificar recursos, priorizar e implementar la transformación
 - Oportunidades y soluciones
 - Plan de migración
- Gobierno de la arquitectura: Monitoreo y gestión de la arquitectura para que se mantenga alineada con el negocio
 - Implementación de gobierno
 - Mantenimiento y cambio de la arquitectura



GOBIERNO DETI



GOBIERNO DETI: COBIT

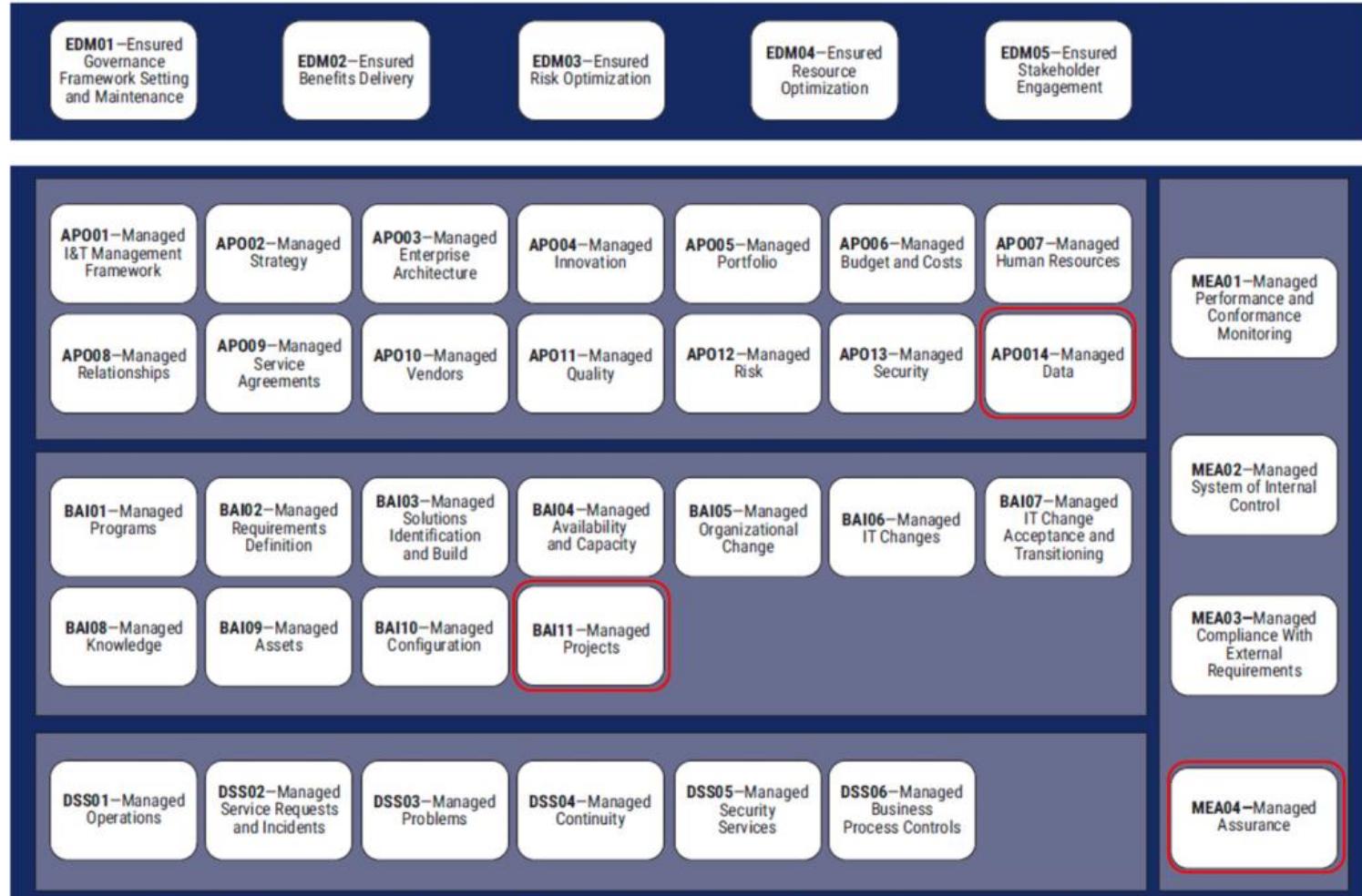
Andrés Armando Sánchez Martín



COBIT[®]2019

 **ISACA**[®]

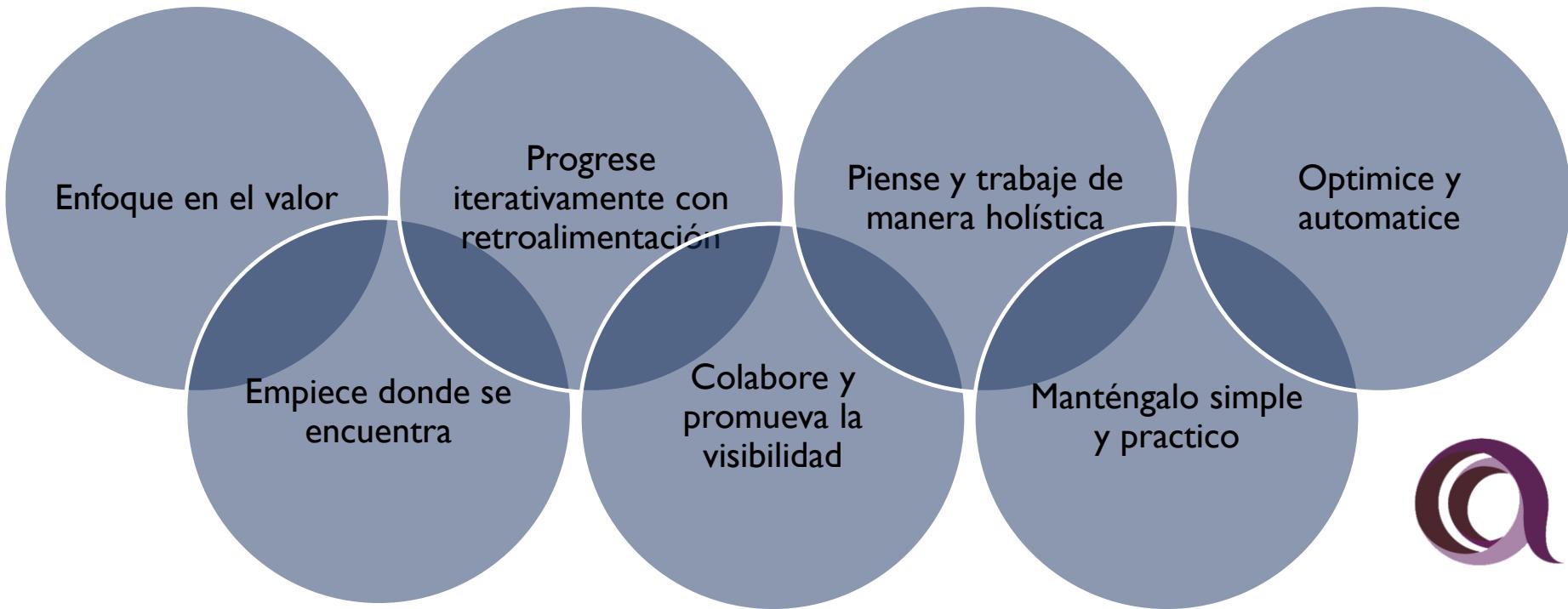
GOBIERNO DETI: COBIT



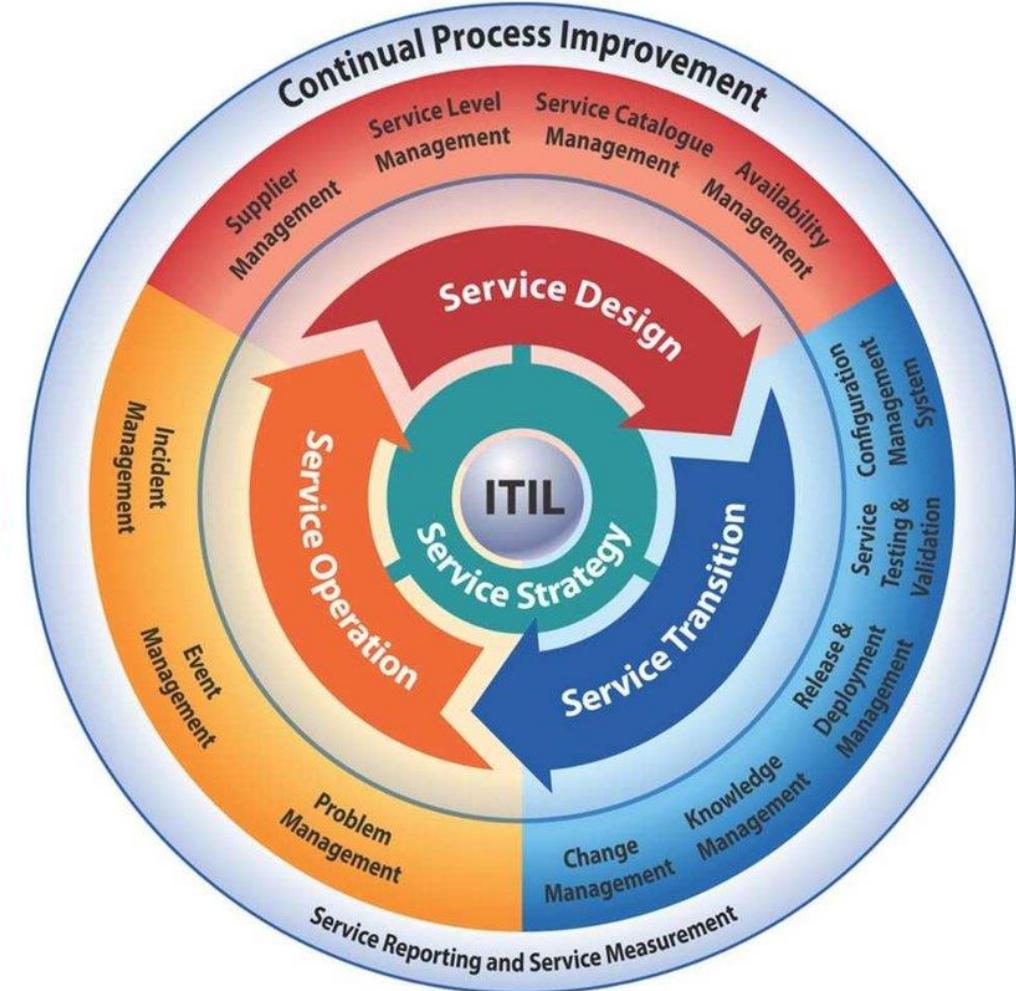
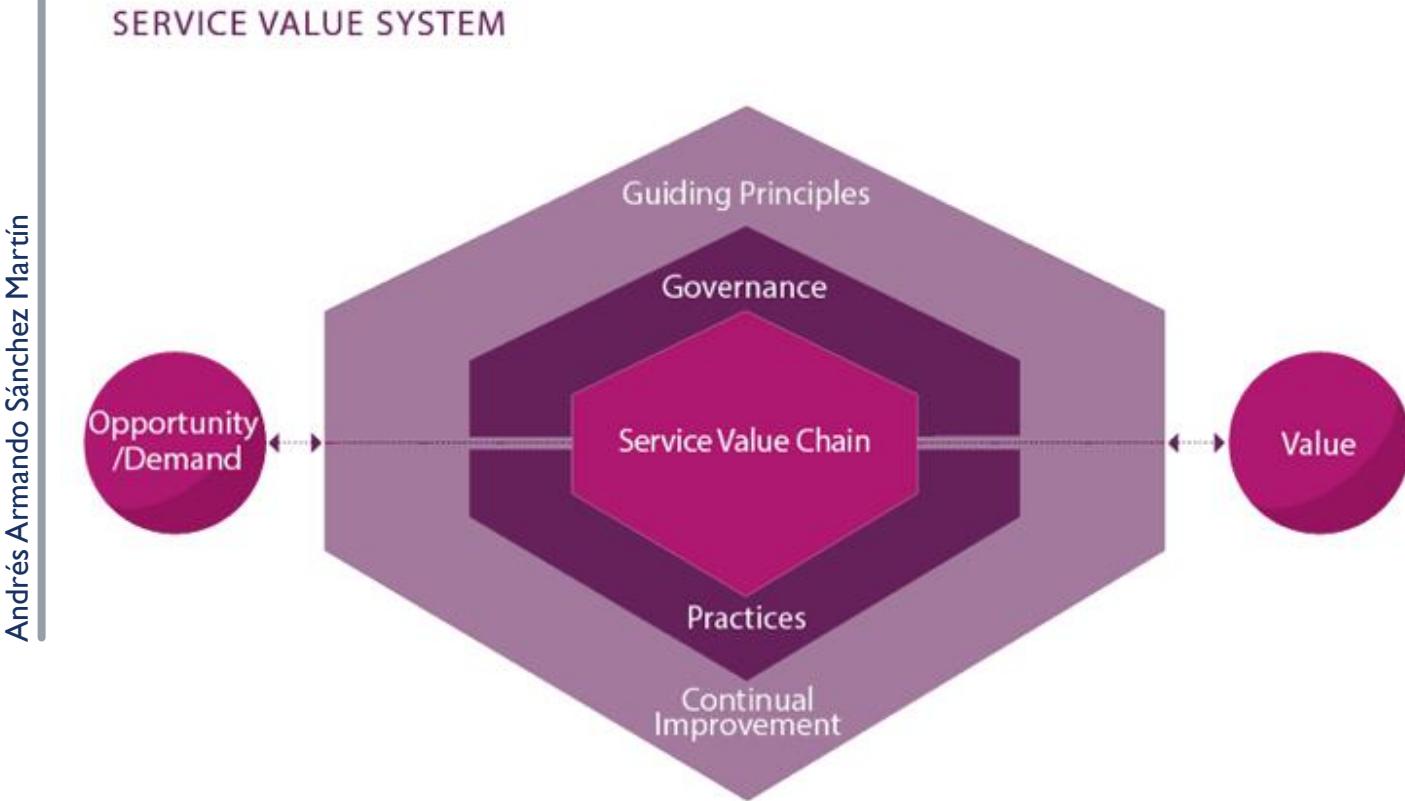
Reference: COBIT® 2019 Framework: Introduction and Methodology, Chapter 4 Basic Concepts: Governance Systems and Components, Figure 4.2

GOBIERNO DETI: ITIL

Andrés Armando Sánchez Martín



GOBIERNO DETI: ITIL



SOFTWARE ENGINEERING INSTITUTE (SEI)

Es un referente en Ingeniería de Software por realizar el desarrollo del modelo SW-CMM (1991) que ha sido el punto de arranque de todos los que han ido formando parte del modelo que ha desarrollado sobre el concepto de capacidad y madurez, hasta el actual CMMI.



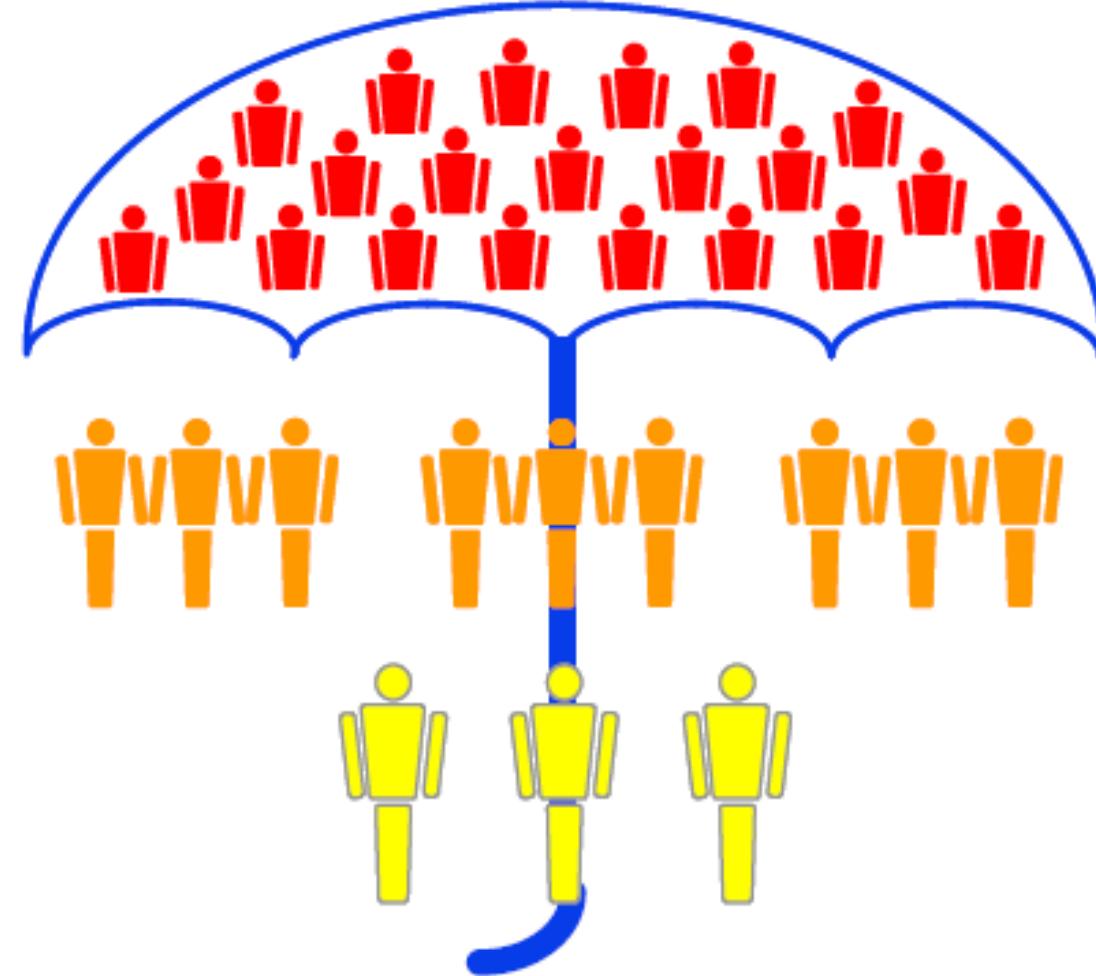
SOFTWARE ENGINEERING INSTITUTE (SEI)

Andrés Armando Sánchez Martín

CMMI - para
capacidad
organizacional

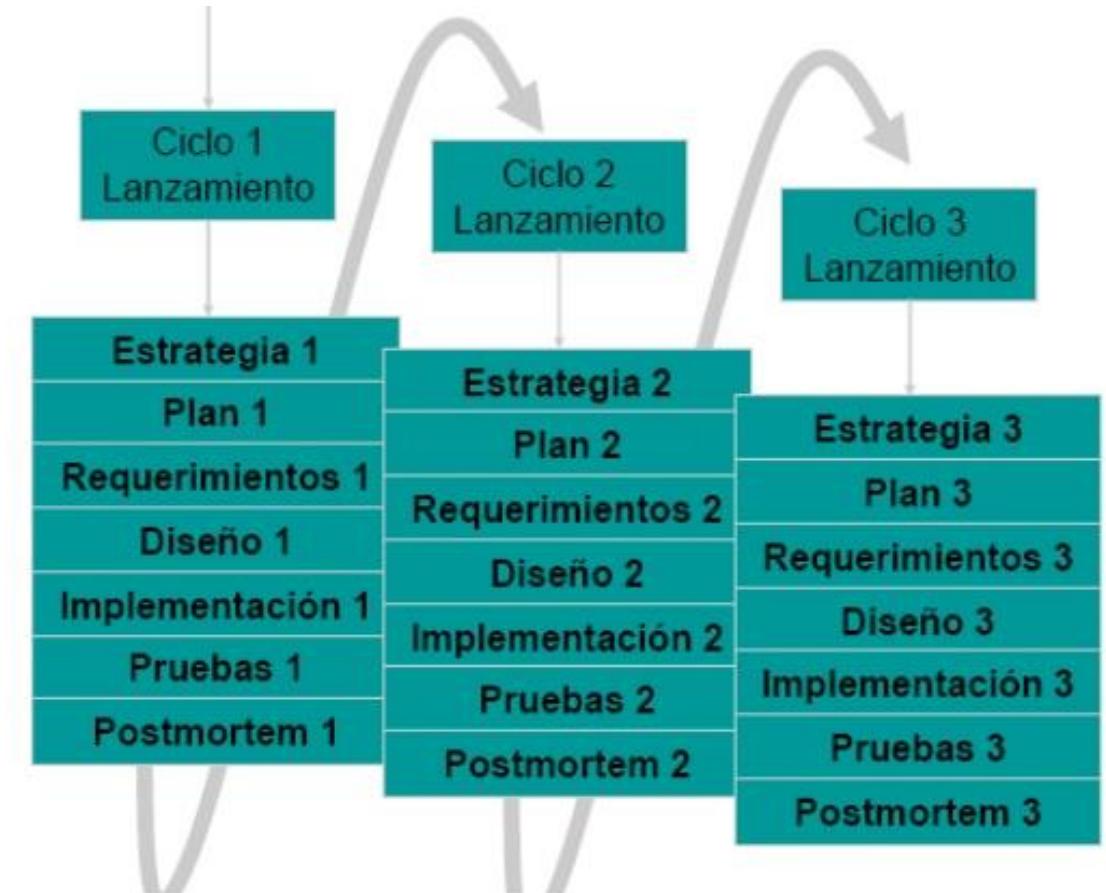
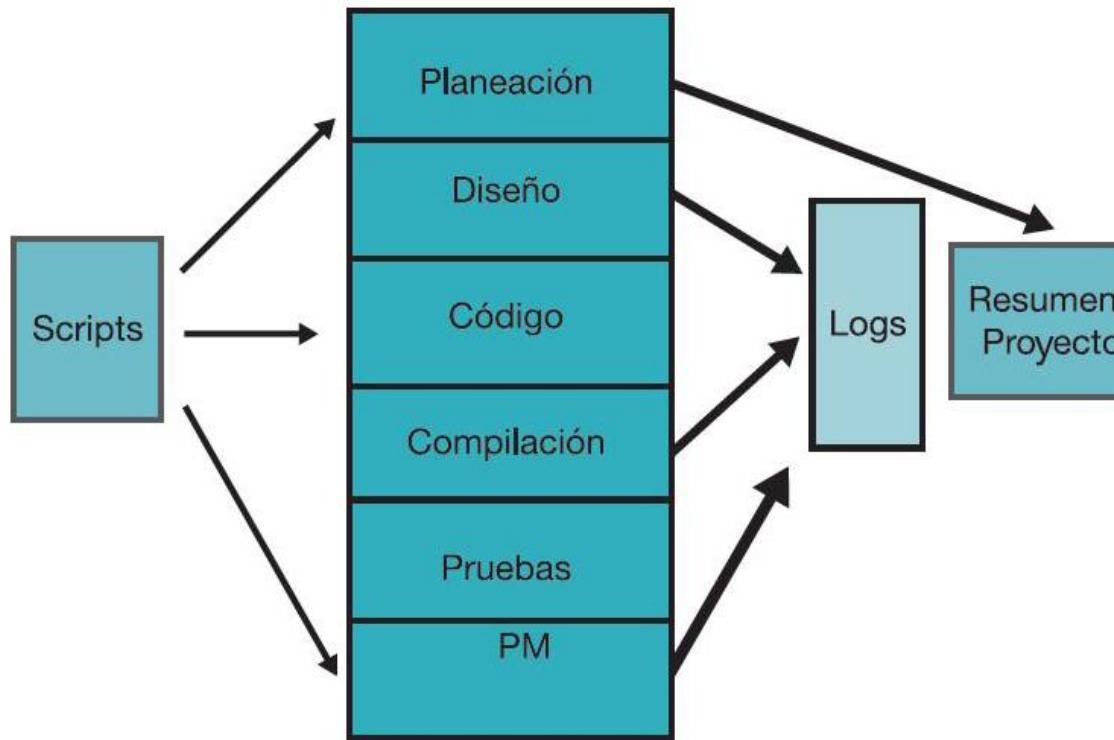
TSP – para
productos con
calidad, a tiempo
y en costo

PSP - para
disciplina y
competencias
individuales

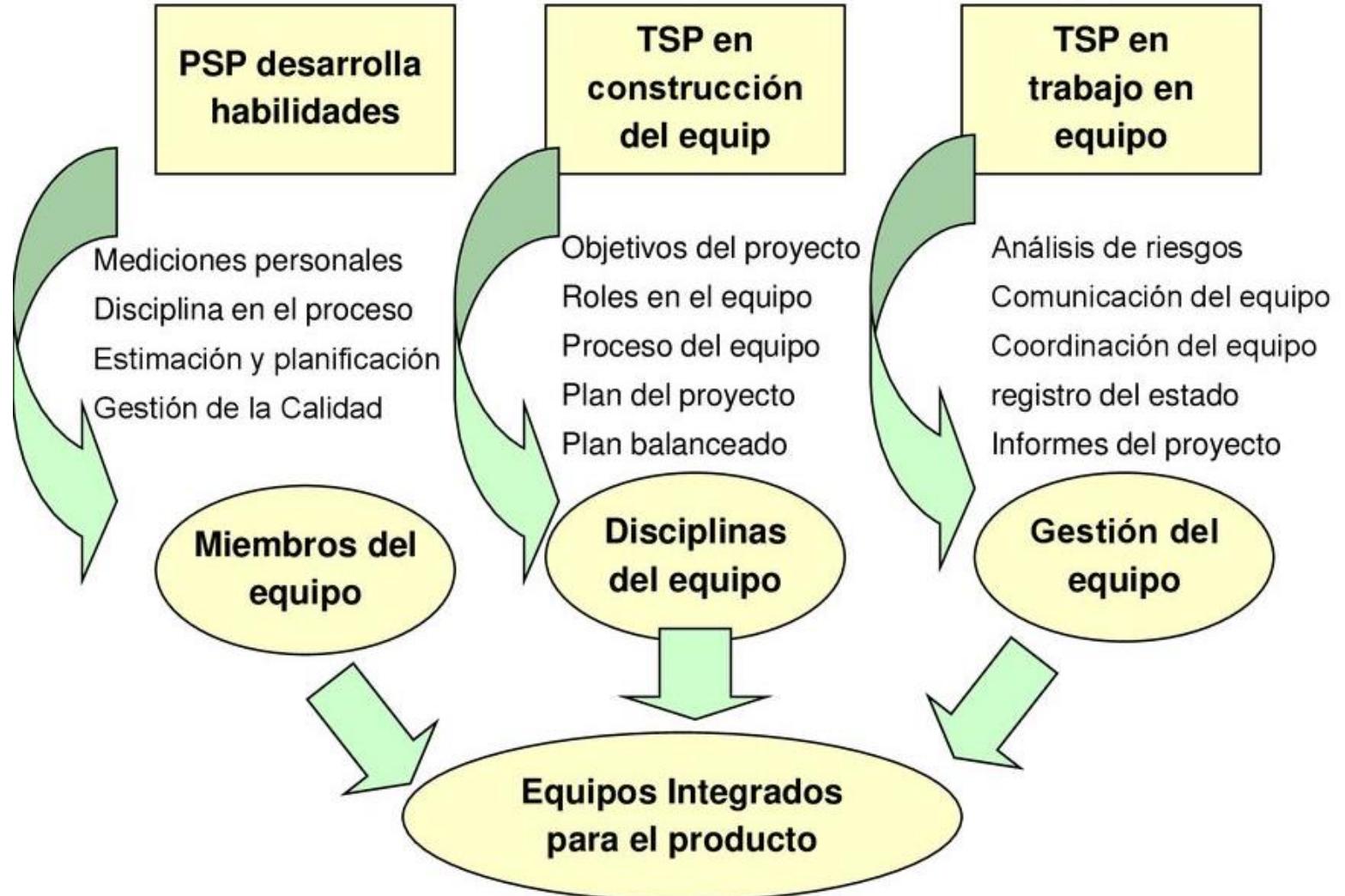


SOFTWARE ENGINEERING INSTITUTE (SEI): PSP Y TSP

Andrés Armando Sánchez Martín



SOFTWARE ENGINEERING INSTITUTE (SEI): PSP Y TSP



SOFTWARE ENGINEERING INSTITUTE (SEI): CMMI



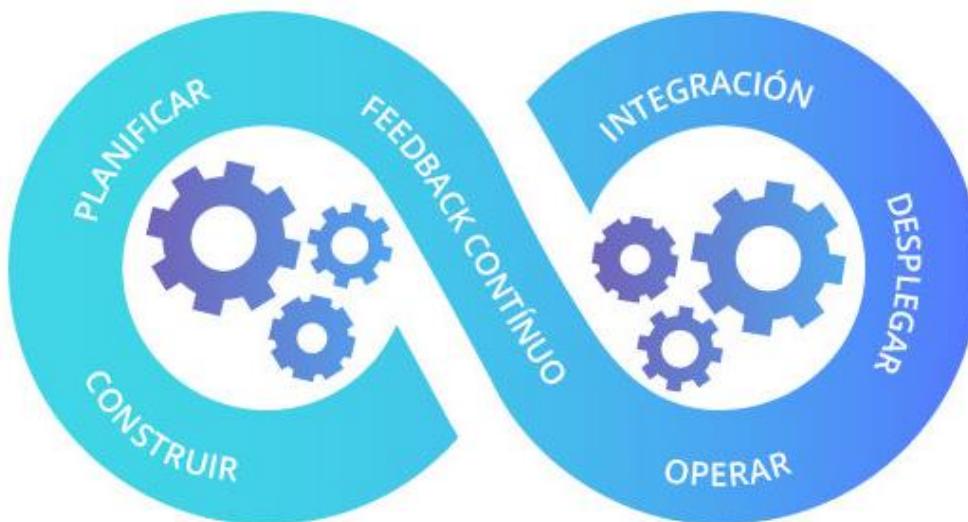
Capability Maturity Model Integration



METODOLOGÍAS AGILES

Manifiesto Ágil

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

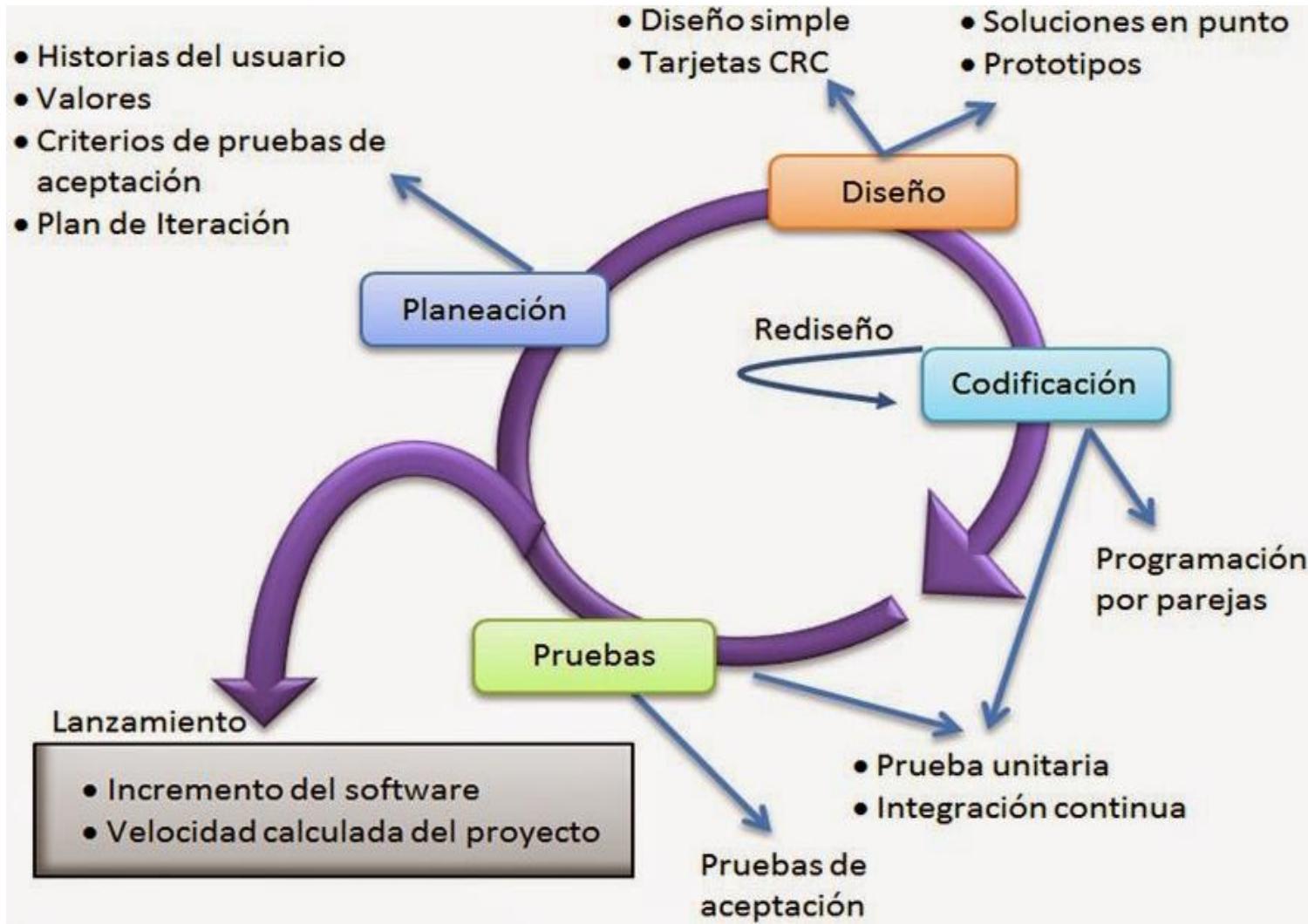


METODOLOGÍAS AGILES: XP

Valores

- Simplicidad
- Comunicación
- Retroalimentación (feedback)
- Coraje o valentía
- Respeto

eXtreme Programming



METODOLOGÍAS AGILES: SCRUM

Propietario del producto

Scrum Master

Equipo de Scrum

Socios

Vendedores

Cuerpo de asesoramiento de Scrum

Principios

- Control de proceso empírico
- Auto organización
- Colaboración
- Priorización basada en valor
- Tiempo asignado
- Desarrollo iterativo

- Clientes
- Usuarios
- Patrocinadores



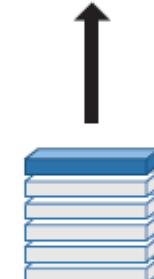
Cronograma de lanzamiento
Sesión de planificación de lanzamiento



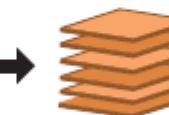
Caso de negocio del proyecto



Declaración de la visión del proyecto
Reunión de visión del proyecto



Backlog Priorizado del Producto



Sprint Backlog
Reunión de planificación del sprint



Entregables aceptados
Reunión de revisión del sprint Reunión de retrospectiva

Daily Standup

Diario
Crear entregables

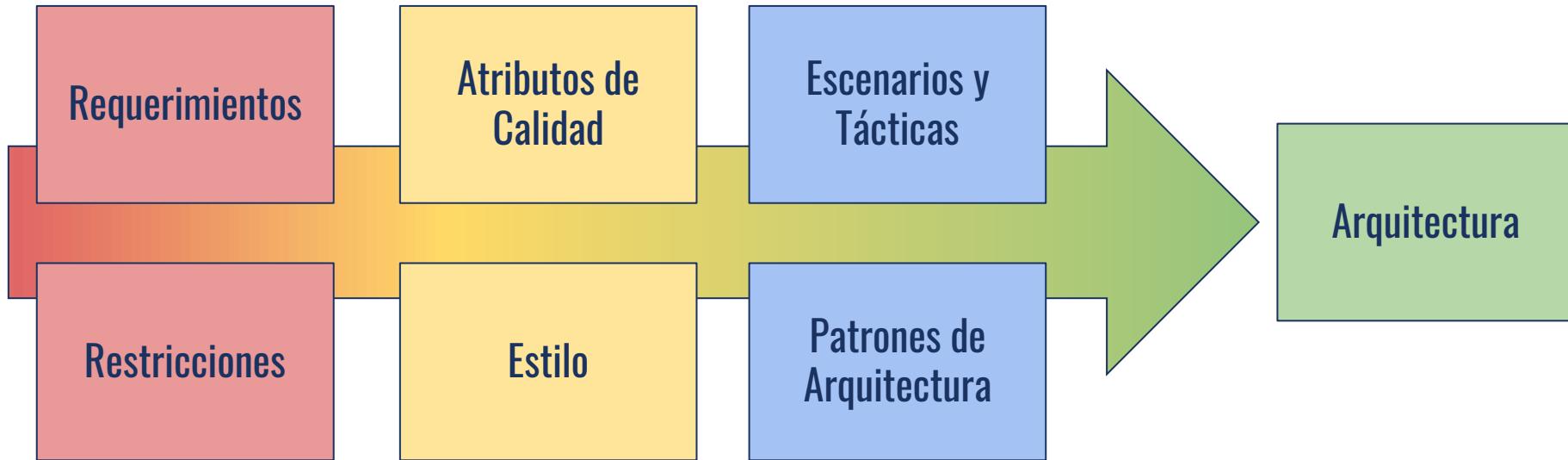
Sprint
1-6 semanas

METODOLOGÍAS AGILES: SCRUM

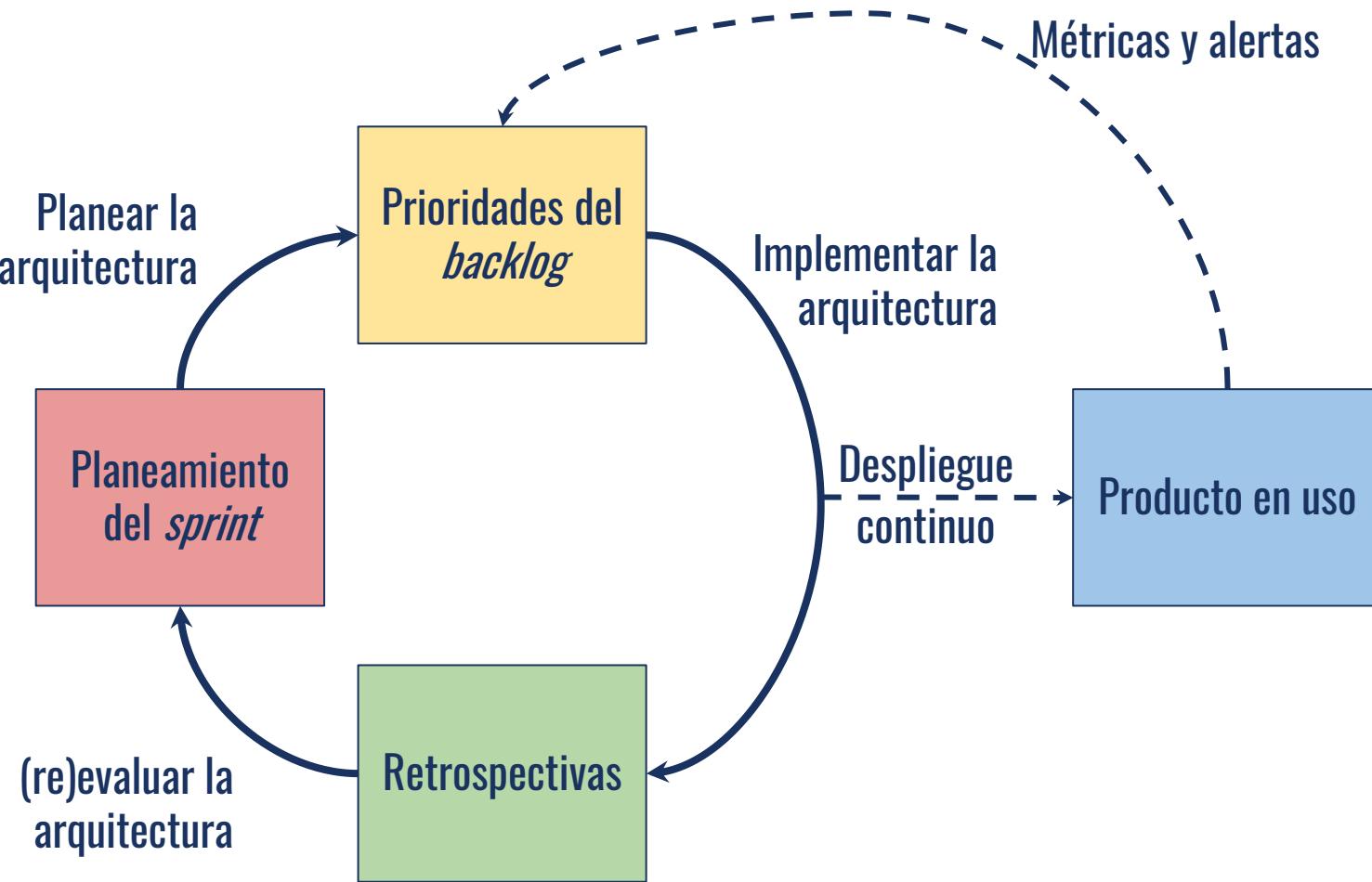
Inicio	Planificación y estimación	Implementación	Revisión y retrospectiva	Lanzamiento
Crear la visión del proyector	Crear historias de usuario	Crear entregables	Demostrar y validar el sprint	Enviar entregables
Identificar al Scrum Master y Stakeholder(s)	Estimar historias de usuario	Realizar el Daily Standup	Retrospectiva del sprint	Retrospectiva del proyecto
Formar el Equipo Scrum	Comprometer historias de usuario	Refinar el Backlog Priorizado del Producto	Reuniones	<ul style="list-style-type: none"> ■ Reunión de visión del proyecto ■ Reunión de planificación del lanzamiento ■ Reunión de planificación del Sprint ■ Reunión diaria de Standup ■ Reunión de revisión del Sprint ■ Reunión de retrospectiva del Sprint
Desarrollar épicas	Identificar tareas			
Crear el Backlog Priorizado del Producto	Estimar tareas			
Realizar la planificación del lanzamiento	Crear el Sprint Backlog			

ENFOQUE TRADICIONAL

Andrés Armando Sánchez Martín

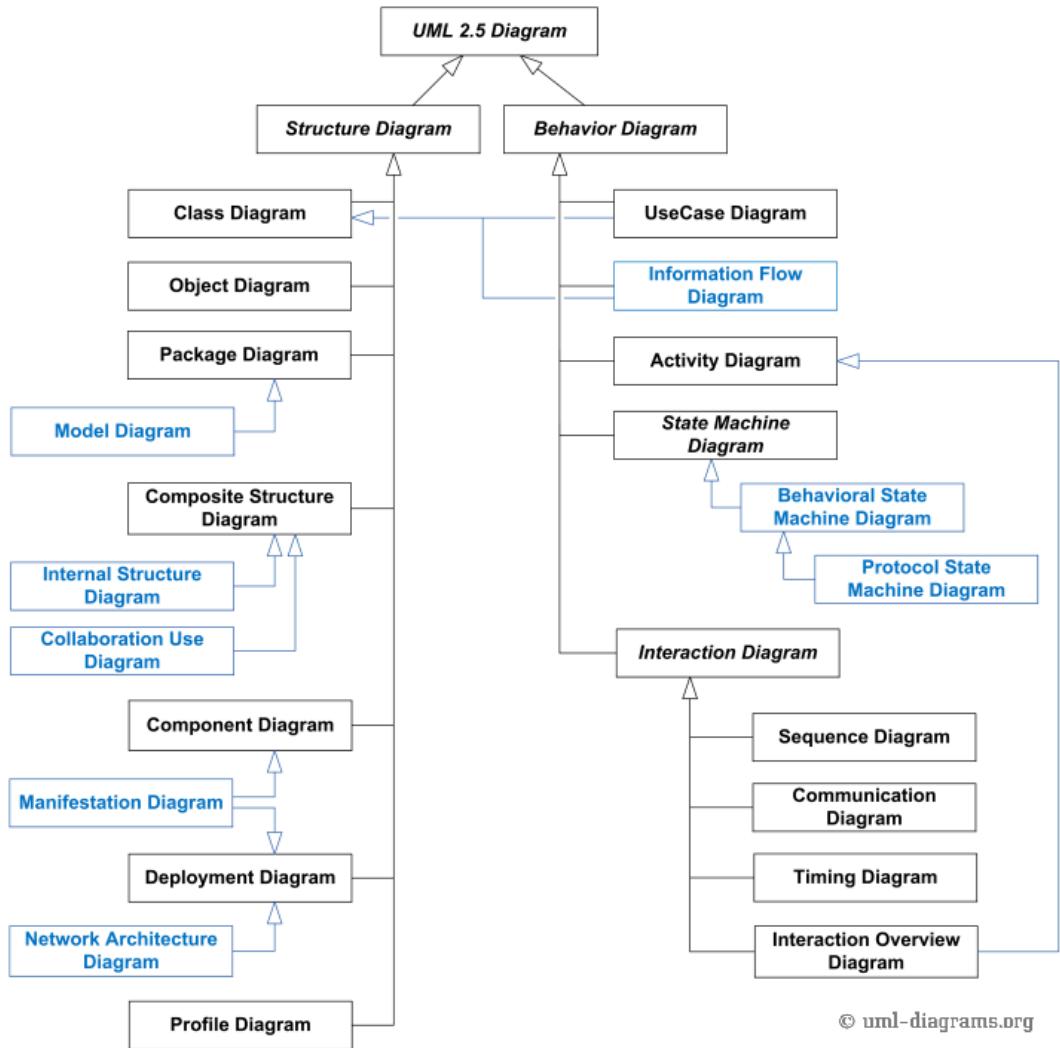


ENFOQUE AGIL



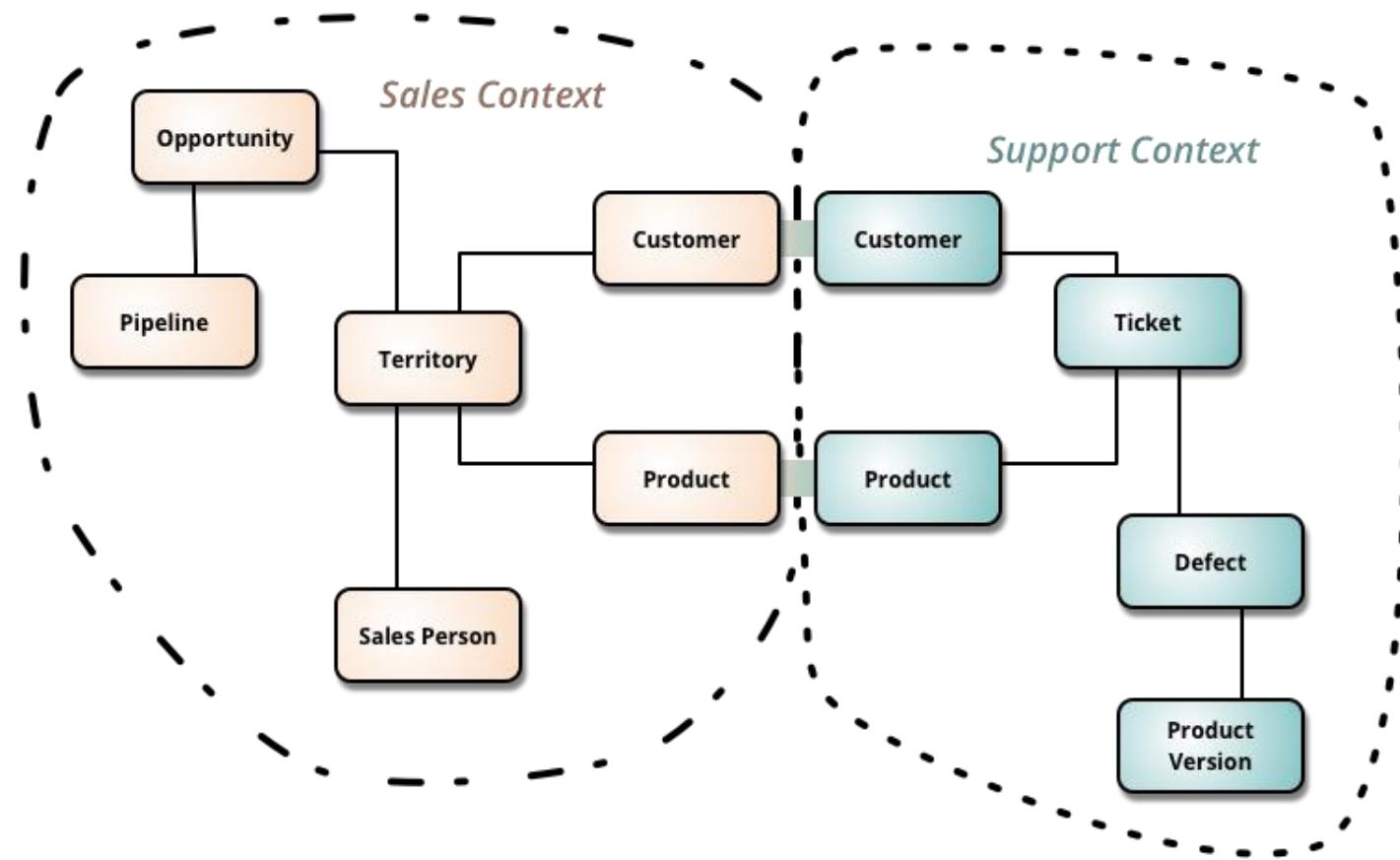
LENGUAJE UNIFICADO DE MODELADO - UML

2.5	mayo 2015
2.4.1	julio 2011
2.3	mayo 2010
2.2	enero 2009
2.1.2	octubre 2007
2.0	julio 2005
1.5	marzo 2003
1.4	septiembre 2001
1.3	febrero 2000
1.2	julio 1999
1.1	diciembre 1997

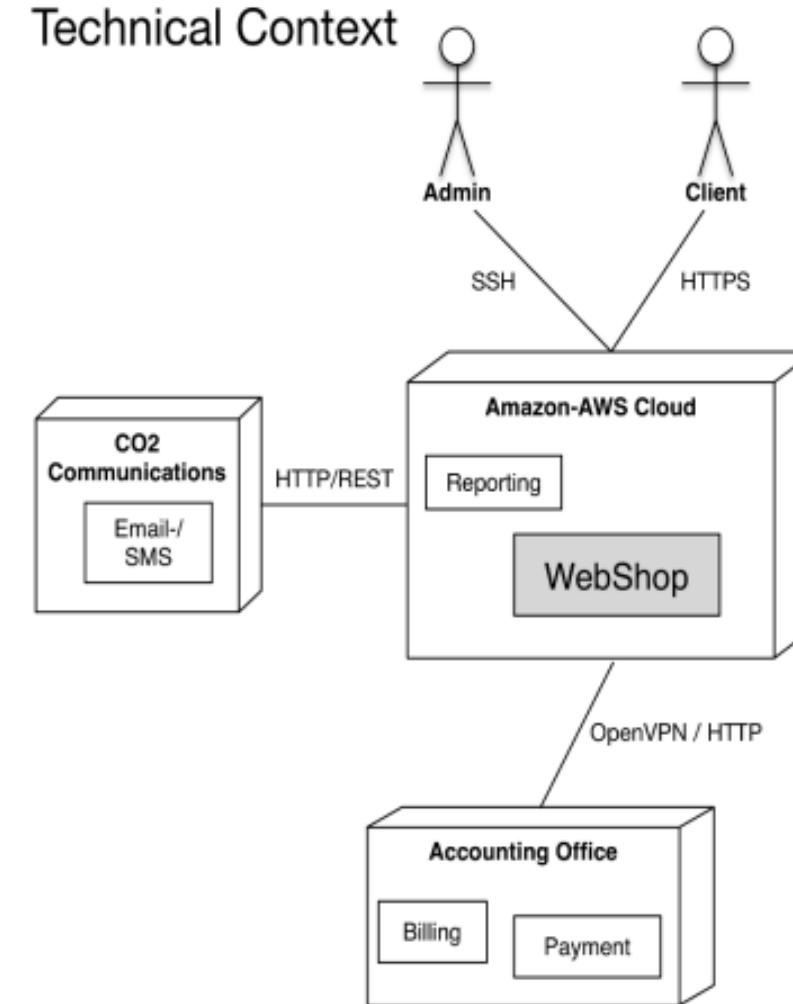
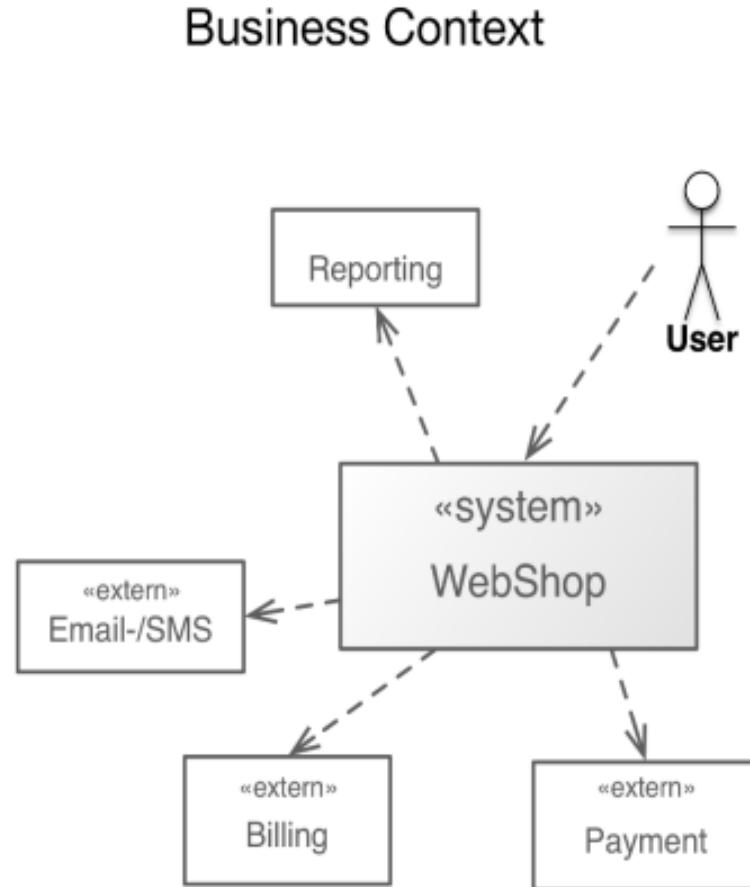


DDD - DOMAIN-DRIVEN DESIGN

- colocar el enfoque principal del proyecto en el dominio central y la lógica del dominio
- basar diseños complejos en un modelo del dominio
- iniciar una colaboración creativa entre expertos técnicos y de dominio para refinar iterativamente un modelo conceptual que aborde problemas de dominio particulares.



DDD - DOMAIN-DRIVEN DESIGN



BASADO EN VISTAS

- Seleccionar un conjunto de puntos de vista
 - De acuerdo a las necesidades de los stakeholders
- Definir vistas de acuerdo a los puntos de vista
- Añadir un documento "Extra"
 - Arquitectura general
 - Información sobre cómo se relacionan las vistas

El propósito de las vistas y los puntos de vista es permitir que los involucrados comprendan sistemas muy complejos, organicen los elementos del problema y la solución en torno a dominios de experiencia, los puntos de vista a menudo corresponden a capacidades y responsabilidades dentro de la organización.

- Varias posibilidades
 - Vistas 4+1 de Kruchten
 - Vistas y más
 - Modelo C4
 - Plantillas Arc42

BASADO EN VISTAS: VISTAS Y PUNTOS DE VISTA

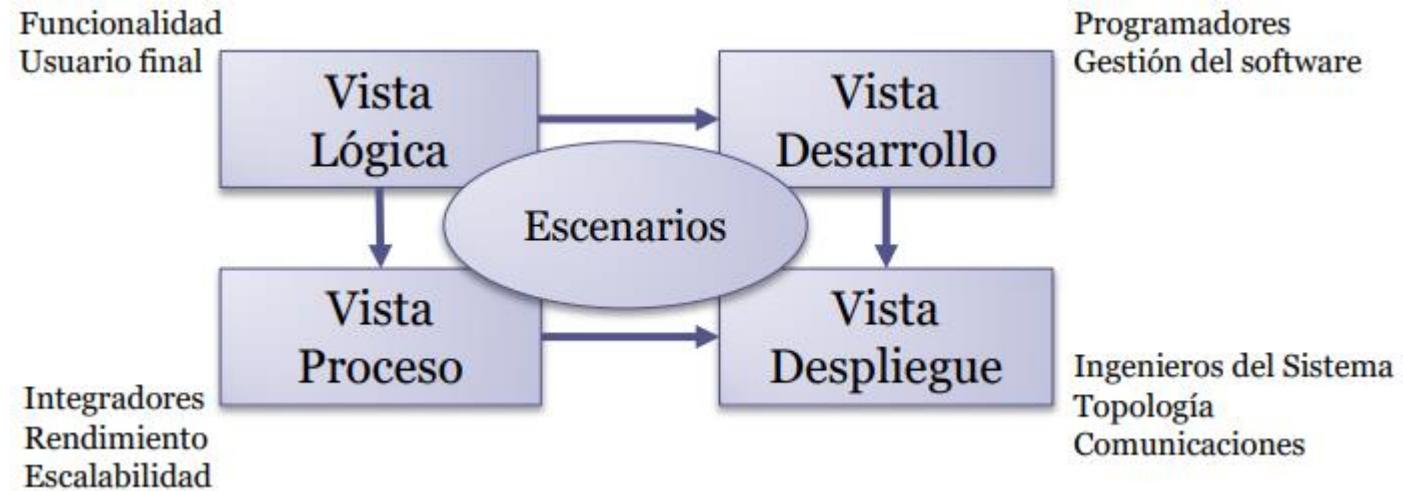
- Arquitectura del software = entidad compleja
 - No puede describirse con una dimensión simple
 - Requiere varias vistas para diferentes stakeholders
- Vista = Representación de un sistema respecto a algunas preocupaciones
 - Diferentes vistas sirven para diferentes objetivos
- Punto de vista = Colección de patrones, plantillas y convenciones para construir una vista
 - Ejemplos: estructura, comportamiento, despliegue

Una vista es lo que ves

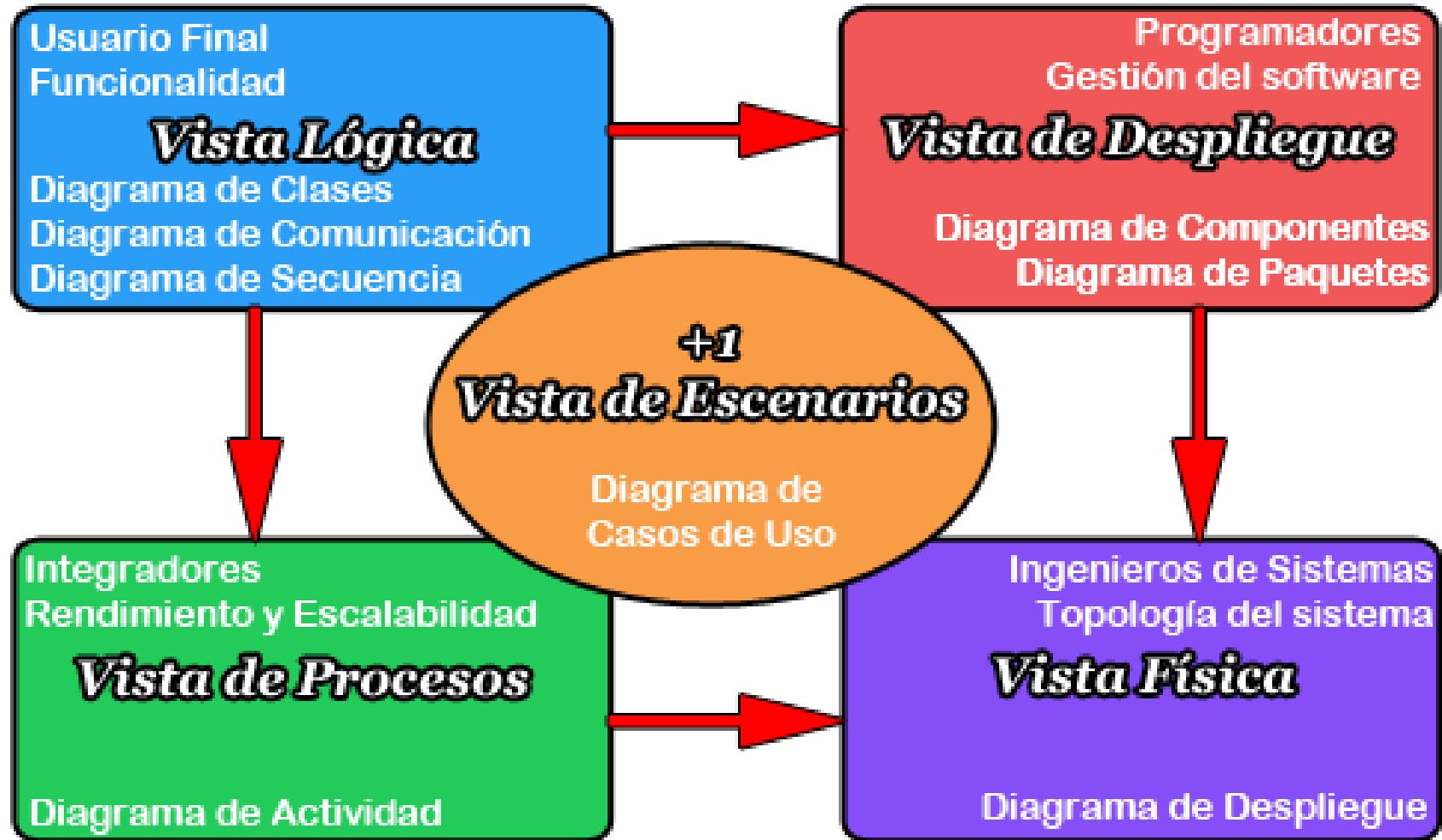
Un punto de vista es desde donde estas mirando

BASADO EN VISTAS: VISTAS 4+ IDE KRUCHTEN

- Adoptadas en el Rational Unified Process 5 vistas:
 - 1 Vista Lógica: funcionalidad del sistema
 - 2 Vista de Desarrollo: módulos, capas,...
 - 3 Vista de Proceso: unidades de ejecución, concurrencia...
 - 4 Vista Física: Topología de infraestructura y desarrollo
 - (+1) Vista de Escenarios: casos de uso o escenarios



BASADO EN VISTAS: VISTAS 4+ IDE KRUCHTEN



BASADO EN VISTAS: MODELO C4

Se basa en describir:

- **Contexto:** Diagrama de contexto o empresarial
- **Contenedores:** Estructura de alto nivel
- **Componentes:** zoom y descomponer
- **Código:** Diagramas de clases UML, diagramas ER, ...

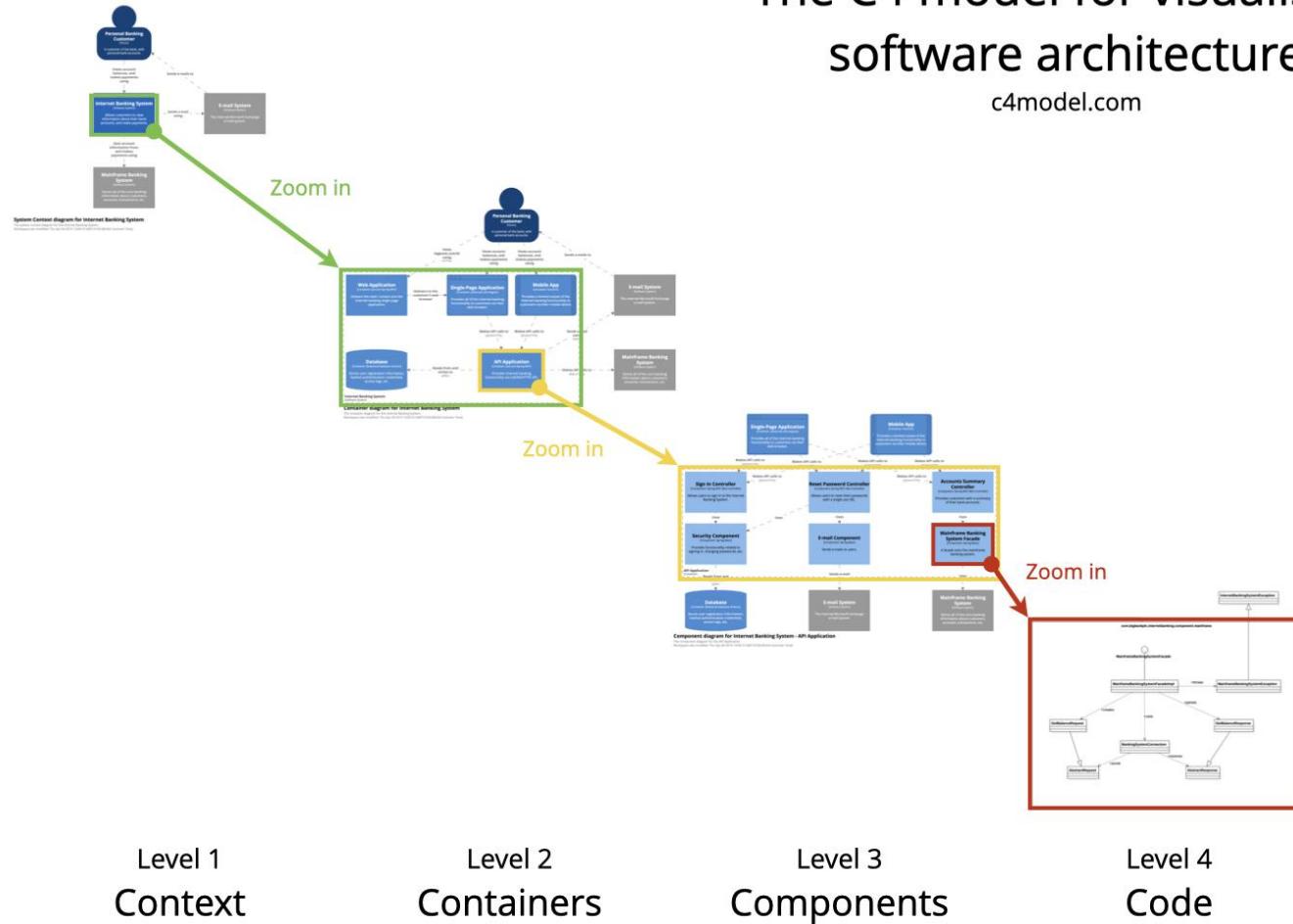
Guía de documentación:

- Contexto
- Resumen funcional
- Atributos de calidad
- Restricciones
- Principios
- Código
- Datos
- Infraestructura de la arquitectura
- Despliegue
- Entorno de desarrollo
- Operaciones y soporte
- Log de decisiones

BASADO EN VISTAS: MODELO C4

The C4 model for visualising
software architecture

c4model.com



ESTRUCTURA DEL DOCUMENTO DE DISEÑO DE SOFTWARE (SDD): INTENCIÓN

- **Objetivo principal:** Comunicar la estructura
 - Comprender la visión general del sistema
- **Crear una visión compartida:** equipo y otras personas interesadas
 - Vocabulario común
- **Describir** qué software se está construyendo y cómo
 - Facilitar conversaciones técnicas sobre características
 - Proporcionar mapas para navegar el código fuente
- **Justificar** decisiones de diseño
- **Ayudar** a desarrolladores nuevos que se unen al equipo

ESTRUCTURA DEL DOCUMENTO DE DISEÑO DE SOFTWARE (SDD): REGLAS

- Escribir desde el punto de vista del lector
 - Encontrar quienes serán los lectores y sus expectativas
- Evitar repeticiones innecesarias (**principio DRY**)
- Evitar ambigüedad
 - Explicar la notación (o utilizar notaciones estándar)
 - Utilizar leyendas o claves para diagramas
- Utilizar una organización estándar o plantilla
 - Añadir Pendiente (TBD/To do) cuando sea necesario
 - Organizar para referencias/enlaces rápidos
- Registrar las justificaciones de las decisiones
- Mantener la documentación actual

ESTRUCTURA DEL DOCUMENTO DE DISEÑO DE SOFTWARE (SDD): REQUISITOS

- Comprendible por las diferentes personas interesadas
 - Stakeholders técnicos y no-técnicos
- Reflejar la realidad
 - Cuidado con separación modelo-código (model-code gap)
- Adaptarse a cambios
 - Adaptarse a proyectos ágiles
 - Arquitectura evolutiva

ESTRUCTURA DEL DOCUMENTO DE DISEÑO DE SOFTWARE (SDD): DOCUMENTANDO LAS VISTAS

- Introducción
 - Descripción textual de la vista
- Diagrama(s)
 - Añadir título descriptivo incluyendo estructuras representadas
 - Crear una leyenda explicando significado de símbolos
 - No olvidarse de explicar líneas/flechas
- Lista de elementos y responsabilidades
 - Dar nombres descriptivos
 - Definir términos (inclusión de glosario)
- Justificación de decisiones
- Tratar de conseguir consistencia y simplicidad
Mantener los elementos consistentes
 - Colores, formas, flechas,...
 - Si se utiliza un esquema de color, seguirlo consistentemente
 - Chequear nombres entre vistas
 - Registrar posibles inconsistencias
- Evitar demasiados detalles
 - Recordar ley de Miller
 - Persona media puede memorizar 7 (± 2) elementos

ESTRUCTURA DEL DOCUMENTO DE DISEÑO DE SOFTWARE (SDD)

- Portada
- Control de Versiones
- Control de Cambios
- Tablas de contenidos, tablas y diagramas
- Introducción
 - Alcance
 - Objetivos
- Requerimientos
 - Identificación de Usuarios
 - Definición de Requerimientos (Historias de usuarios)
 - Atributos de Calidad
 - Restricciones
 - Riesgos
- Definiciones de Arquitectura
 - Modelo de vistas
- Niveles
- Distribución del sistema
- Tecnologías
- Patrones de diseño
- Buenas prácticas y estándares
- Modelos del sistema
 - C4 model o 4+1 view model
- Modelo de Datos
 - Diagrama de datos
 - Diccionario de datos
- Interface de Usuario
 - Líneas de diseño gráfico
 - Componentes gráficos
 - Principios de usabilidad y accesibilidad
 - Navegación
 - Mockup





BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué es la ingeniería de requerimientos?
- ¿Qué son: Epics, Features, Stories, Task, Spike, Bug, and Test?
- ¿Cuáles son los criterios de aceptación?
- ¿Culés son las condiciones para la arquitectura?





(1306)

ARQUITECTURA DE SOFTWARE

6.2 Proceso de Arquitectura de Software 2

Agenda

- Documentado la Arquitectura
- Ingeniería de Requerimientos
- Condiciones para la arquitectura





“ESENCIALMENTE,
TODO MODELO ES
INCORRECTO. PERO
ALGUNOS SON
ÚTILES.”

Arquitectura
restrictiva

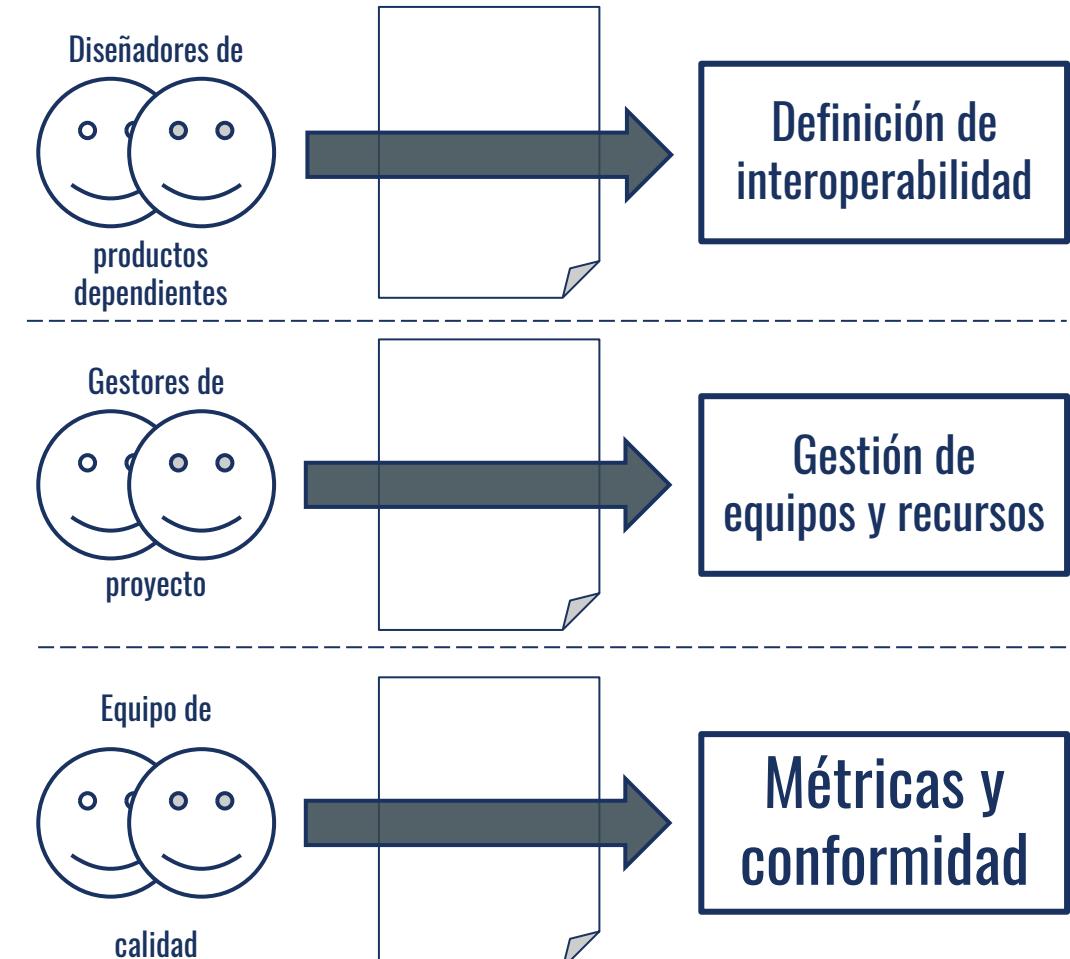
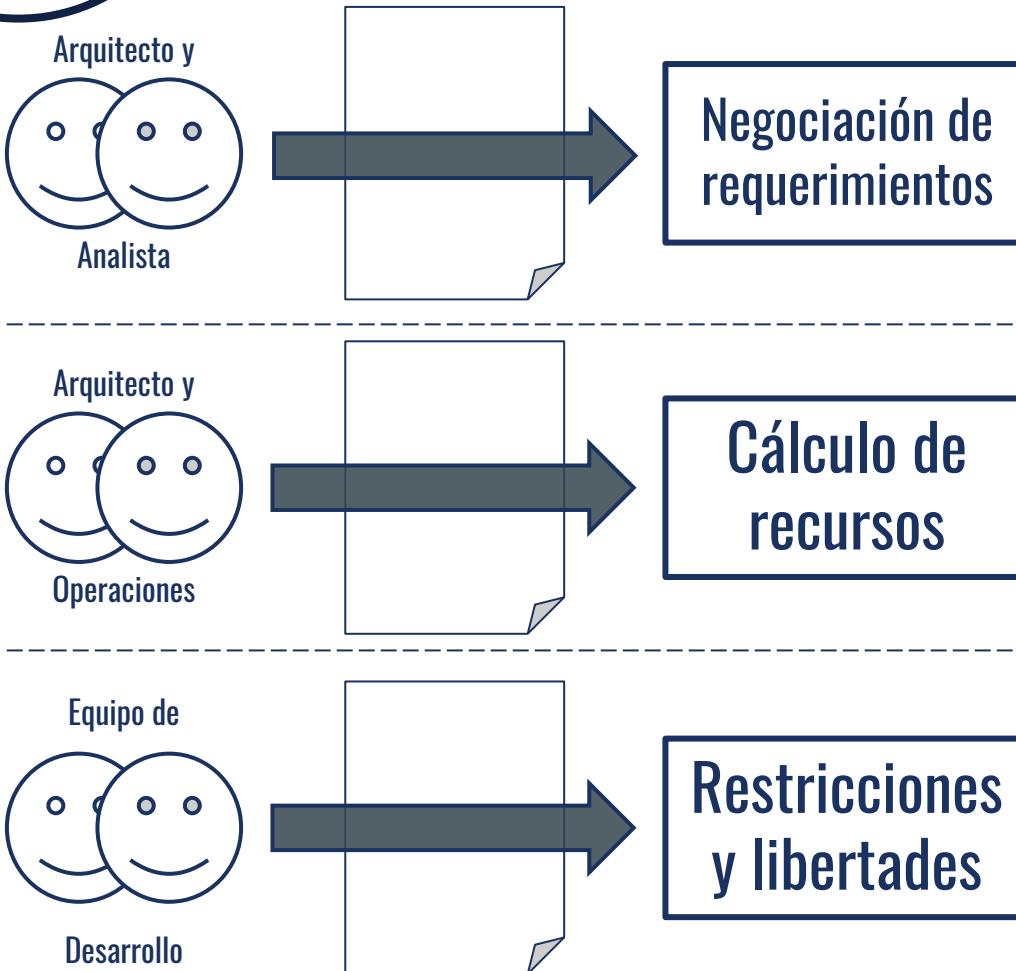
Restringe
las
decisiones
a tomar.

Arquitectura
descriptiva

Documenta
las
decisiones
tomadas.

Empirical Model-Building and Response Surfaces (George Box, 1987)

DOCUMENTO DE ARQUITECTURA



EVOLUCIÓN DEL CÓDIGO

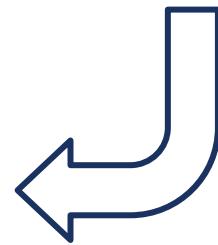
Andrés Armando Sánchez Martín

Modelo de arquitectura

Módulos, componentes, conectores, restricciones, estilo, patrones, atributos de calidad.

Código fuente

Paquetes, clases, interfaces, métodos, funciones, parámetros, tipos.



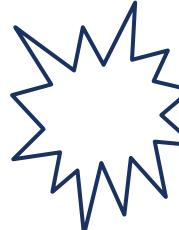
Modelado *ad hoc*



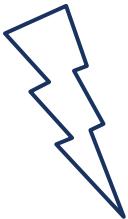
Solo modelos de alto nivel



Sincronización en hitos del ciclo de vida



Sincronización en crisis

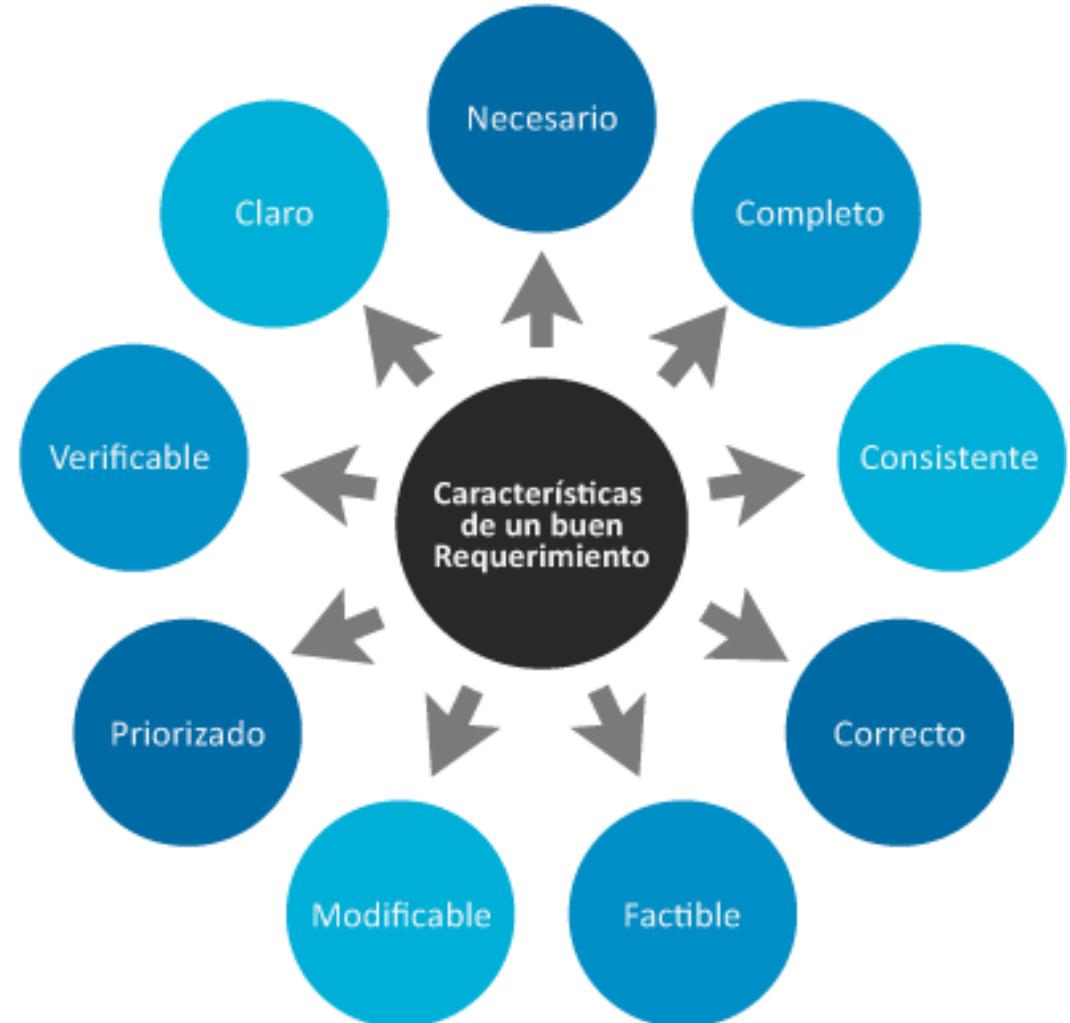


Sincronización constante

Modelo de arquitectura

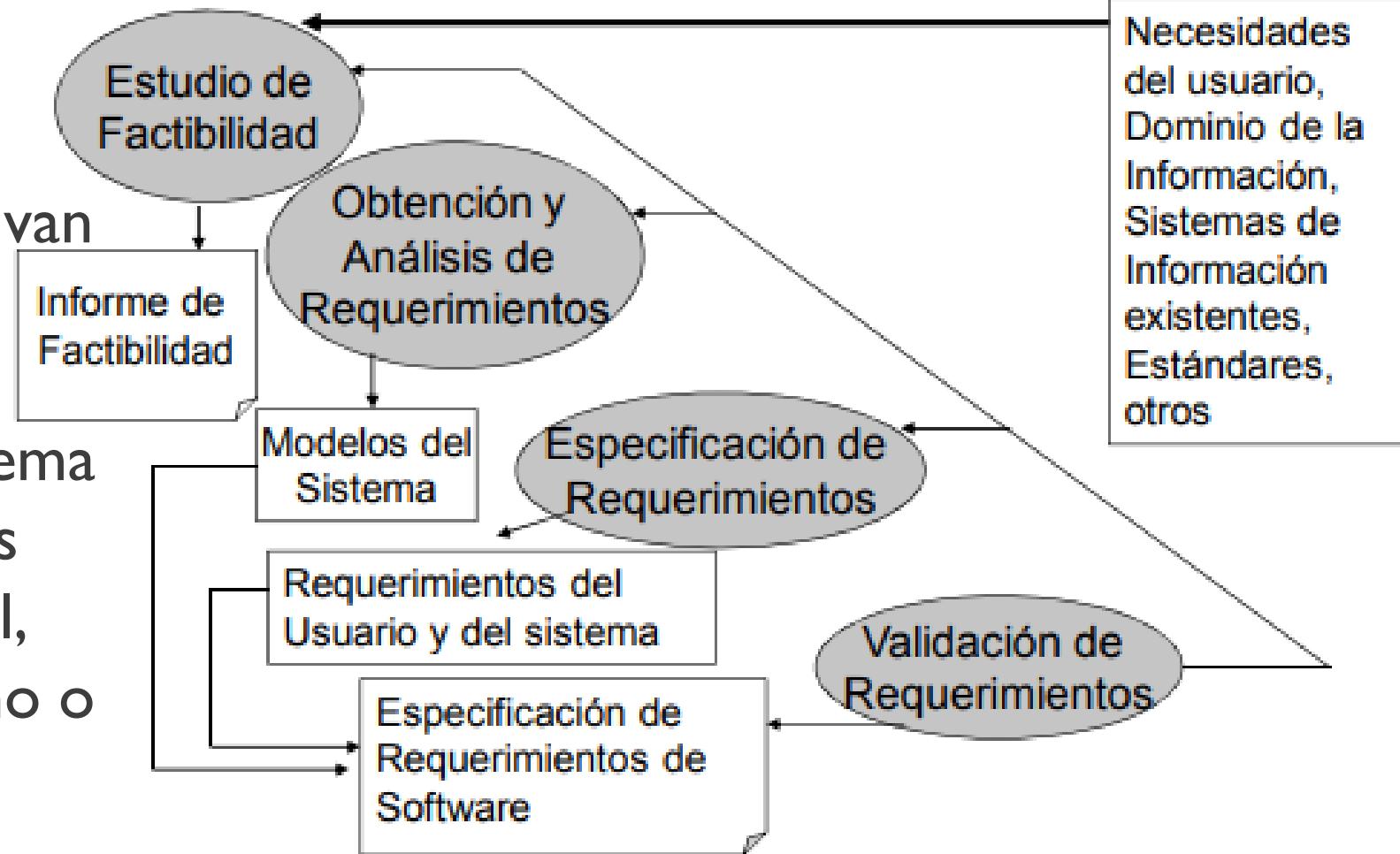
INGENIERÍA DE REQUERIMIENTOS

- Proceso que comprende todas las actividades de necesarias para crear y mantener un documento de requerimientos del sistema.
- Proceso de aplicar un método estructurado, el cual analiza el sistema y desarrolla un conjunto de modelos gráficos del mismo que actúan como una especificación del sistema



INGENIERÍA DE REQUERIMIENTOS: ACTIVIDADES

De las actividades se derivan modelos: El conjunto de modelos describe el comportamiento del sistema al cual se le agregan notas con información adicional, que detallan el desempeño o fiabilidad requeridos.





INGENIERÍA DE REQUERIMIENTOS: ESTRATEGIAS

Entrevistas

Desarrollo Conjunto de Aplicaciones (JAD)

Desarrollo de Prototipos

Observación

Estudio de documentación

Cuestionarios

Tormenta de ideas (Brainstorming)

ETHICS (Implementación Efectiva de Sistemas Informáticos desde los puntos de vista Humano y Técnico)

Puntos de Vista

Escenarios

Etnografía

INGENIERÍA DE REQUERIMIENTOS: BASADO EN PUNTOS DE VISTA

En un sistema existen diferentes tipos de usuarios finales.

Ejemplo: usuarios para un sistema de cajeros automáticos:

- clientes actuales del banco,
- representantes de otros bancos,
- administradores de las sucursales bancarias,
- contadores de la sucursal bancaria,
- administradores de la base de datos,
- administradores de seguridad del banco,
- personas del departamento de marketing
- ing. de mantenimiento de hardware/ software.

- Los enfoques orientados a puntos de vista para la ingeniería de requerimientos toman en cuenta:
 - puntos de vista diferentes y los utilizan para estructurar y organizar tanto el proceso de obtención como los requerimientos mismo.
- Un punto clave del análisis orientado a puntos de vista es que toma en cuenta:
 - la existencia de varias perspectivas y provee un marco de trabajo para descubrir conflictos en los requerimientos propuestos por diferentes usuarios.

BASADO EN PUNTOS DE VISTA: MÉTODO VORD

Se ha diseñado como marco de trabajo orientado a servicios para la obtención y análisis de requerimientos.

Las etapas principales de este método son:

- Identificación de puntos de vista
- Estructuración de puntos de vista
- Documentación de puntos de vista
- Trazado del punto de vista del sistema

MÉTODO VORD: PLANTILLAS

Referencia:	Nombre del punto de vista
Atributos:	Atributos que proveen información del punto de vista
Eventos:	Referencia a un conjunto de eventos que describen cómo reacciona el sistema a eventos del punto de vista
Servicios:	Referencia a un conjunto de descripciones del servicio
Subpuntos de vista:	Nombres de los subpuntos de vista

Referencia:	Nombre del servicio
Fundamento:	Razón del porqué se provee el servicio
Especificación:	Referencia a una lista de especificaciones del servicio. Puede expresarse en diferentes notaciones.
Puntos de vista:	Lista de los nombres de los puntos de vista que reciben el servicio.
Requerimientos no funcionales:	Referencia a un conjunto de requerimientos no funcionales que restringen el servicio
Proveedor:	Referencia a una lista de objetos del sistema que proveen el servicio.

MÉTODO VORD: PLANTILLAS

Referencia: Cliente

Atributos:

- Número de cuenta
- PIN
- Inicio transacción

Eventos:

- Seleccionar servicios
- Cancelar transacción
- Finalizar transacción

Servicios:

- Retiro de efectivo
- Consulta de saldo

Subpuntos de vista:

- Cuentahabiente
- Cliente extranjero

Referencia: Retiro de efectivo

Fundamento:

Mejorar el servicio al cliente y reducir el papeleo

Especificación:

Los usuarios eligen este servicio presionando el botón de retiro de efectivo

Puntos de vista: Cliente

Requerimientos no funcionales:

Entregar efectivo en menos de un minuto de que se ha confirmado la cantidad.

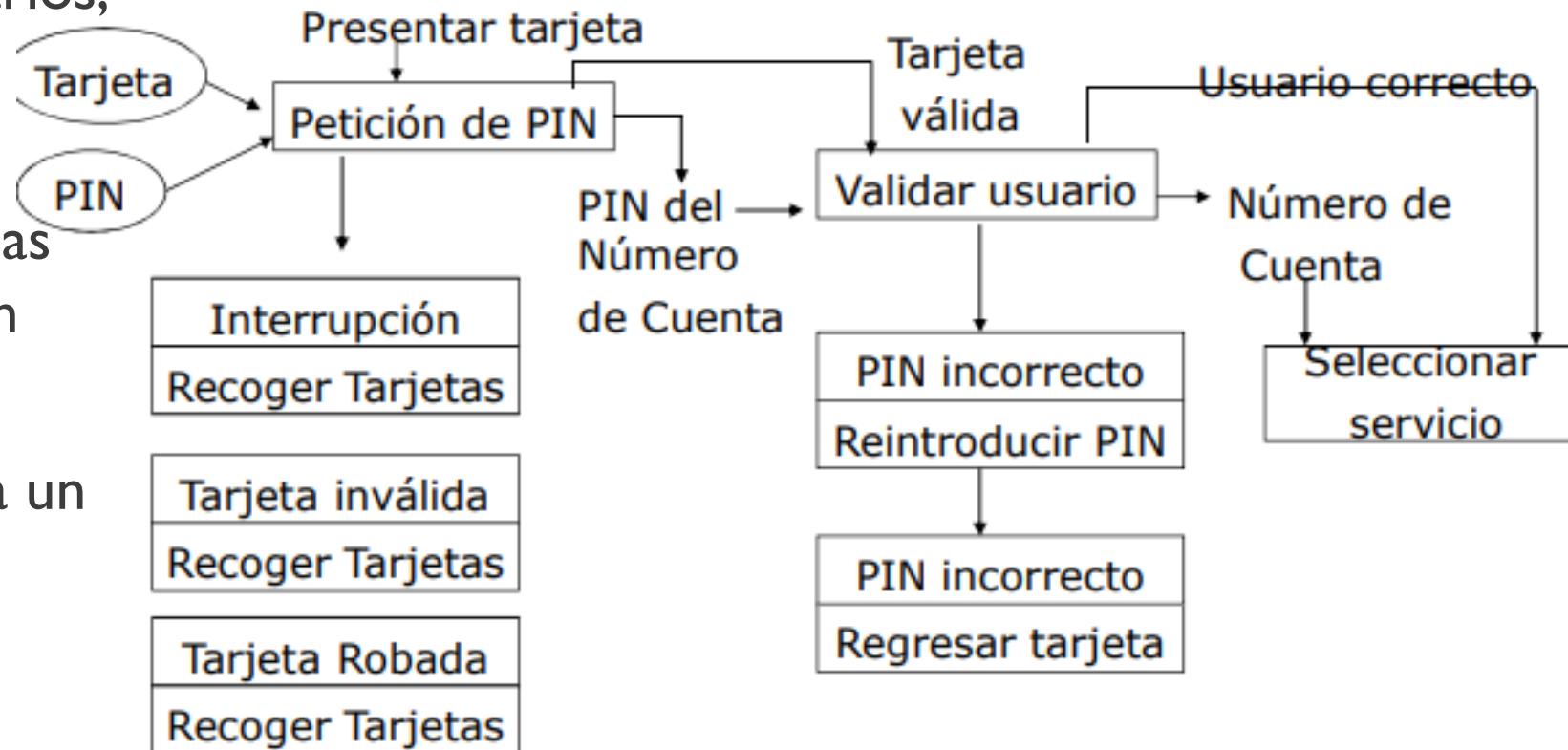
Proveedor: Llenar posteriormente

INGENIERÍA DE REQUERIMIENTOS: BASADO EN ESCENARIOS

- Un escenario empieza con un bosquejo de interacción y se van agregando detalles:
- En general incluyen:
 - Una descripción del estado inicial del sistema
 - Una descripción del flujo normal de eventos
 - Una descripción de lo que puede ir mal y cómo manejarlo
 - Información de otras actividades que se pueden llevar a cabo al mismo tiempo
 - Descripción del estado del sistema al terminar

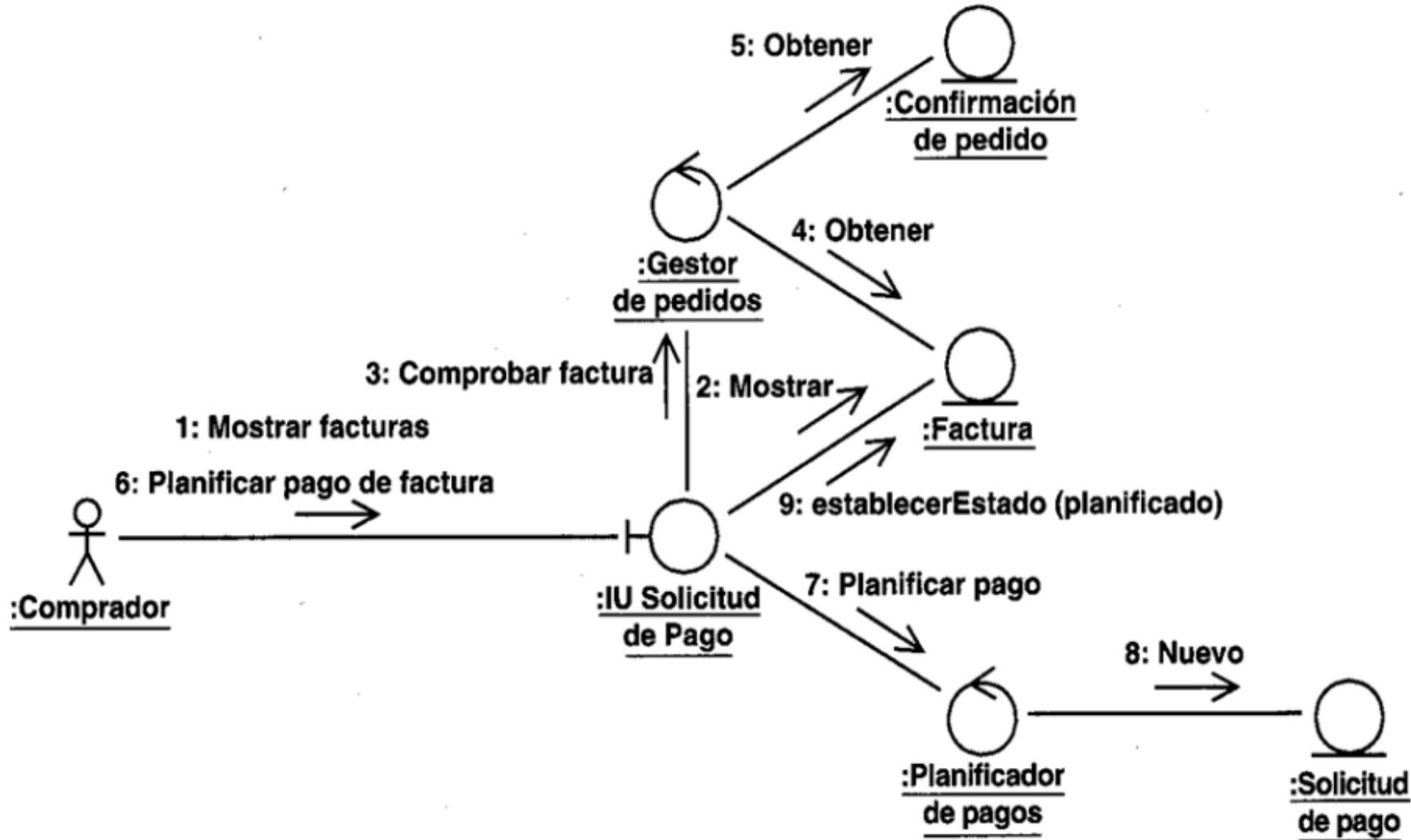
BASADO EN ESCENARIOS: MÉTODO VORD

- Técnica basada en escenarios, propuesta en Objectory [Jacobson, I et al, 1993]
- Parte fundamental de todas las metodologías que usan UML
- Un caso de uso encapsula un conjunto de escenarios



BASADO EN ESCENARIOS: EJEMPLO

Andrés Armando Sánchez Martín



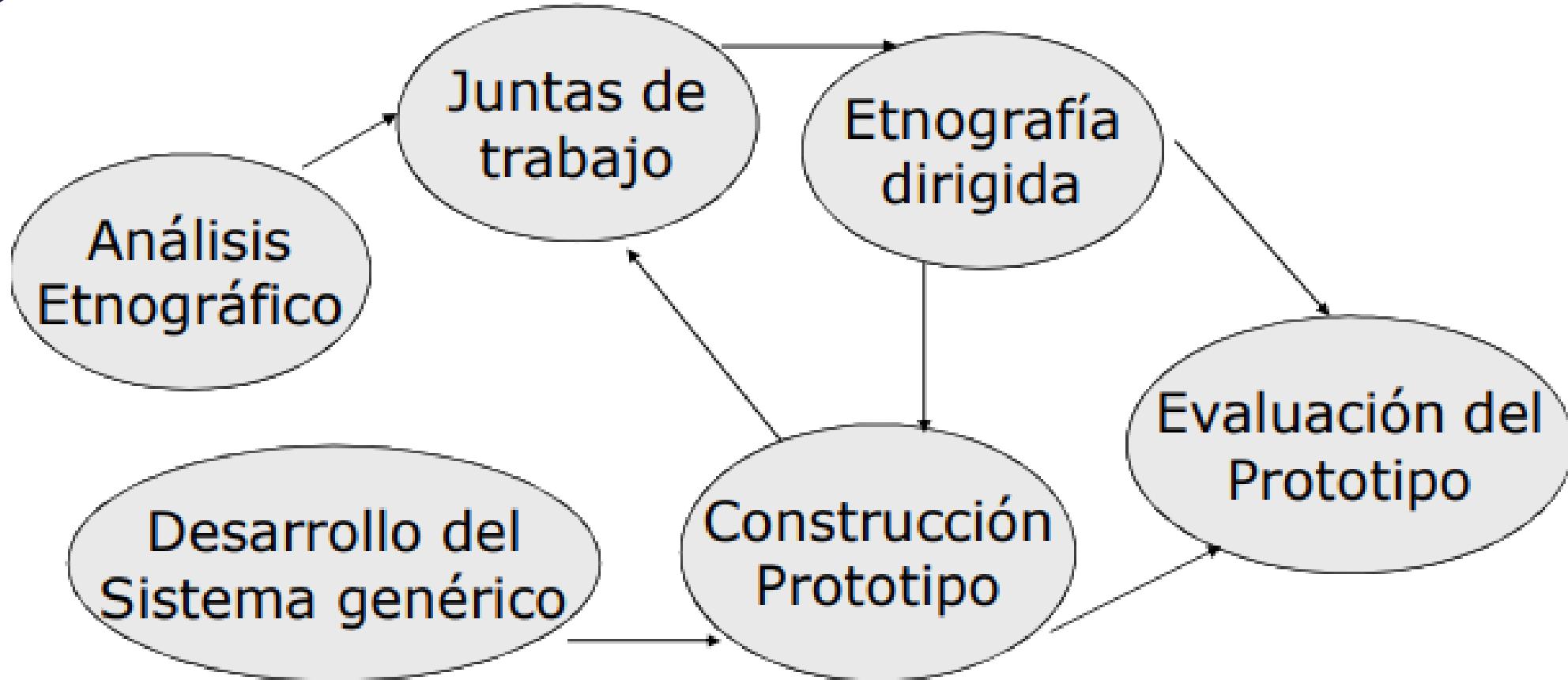


Pontificia Universidad
JAVERIANA
Bogotá

INGENIERÍA DE REQUERIMIENTOS: BASADO EN ETNOGRAFÍA

- Técnica de observación que se puede utilizar para entender requerimientos sociales y organizacionales.
- Especialmente efectiva para descubrir dos tipos de requerimientos.
 - Los derivados de la forma real en que trabajan los usuarios.
 - Los que se derivan del conocimiento y cooperación entre usuarios.
- Combinar con prototipos.
- No sirve para añadir novedades.

BASADO EN ETNOGRAFÍA: ACTIVIDADES



REQUERIMIENTOS EN METODOLOGÍA AGILES

Andrés Armando Sánchez Martín

Criterio de Aceptación
(Acceptance criteria)

Iniciativa (Initiative)

Épica (Epics)

Característica
(Features)

Historia (Story)

Tarea
(Task)

Investigación
(Spike)

Complicaciones
(Bug/Issue)

Pruebas
(Test)

REQUERIMIENTOS EN METODOLOGÍA AGILES: HISTORIAS

Las historias de usuario deben ser:

- **Independientes unas de otras.**
- **Negociables.**
- **Valoradas por los clientes o usuarios.**
- **Estimables.**
- **Pequeñas.**
- **Verificables.**

Historia: Agregar comentarios

Como: Lector del Blog

Quiero: adicionar comentarios a las entradas y recibir alertas cuando otros hagan comentarios

Para: mantenerme en contacto con los demás usuarios del blog

3

Historia: Responder a comentarios

Como: Lector del Blog

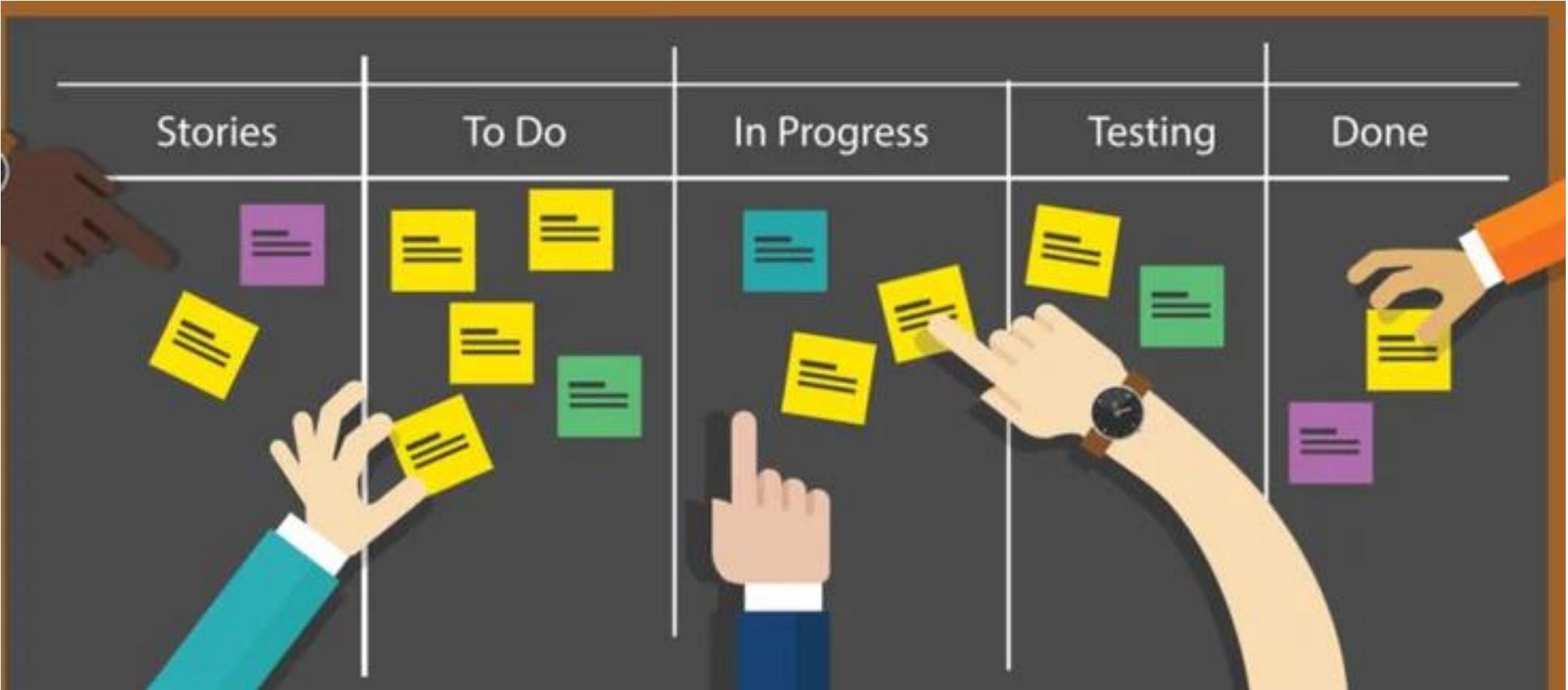
Quiero: adicionar comentarios a las entradas y responder a comentarios de otros lectores

Para: mantenerme en contacto con los demás usuarios del blog

3

SCRUM BOARD

Andrés Armando Sánchez Martín





CONDICIONES PARA LA ARQUITECTURA

Requerimientos

Riesgos

Restricciones

Atributos de calidad

Acuerdos
(trade-off)

Estilos

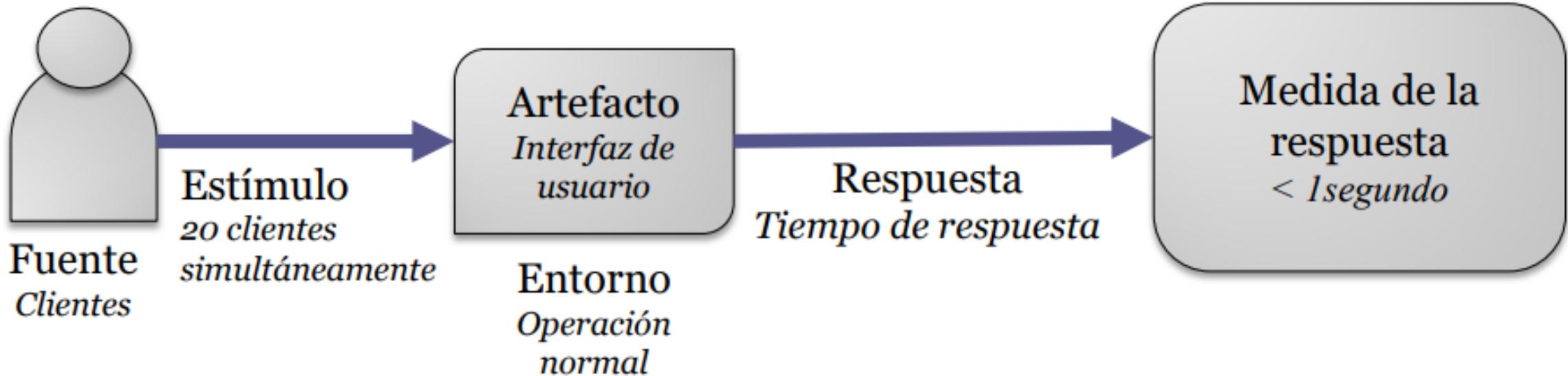
Patrones

Escenarios

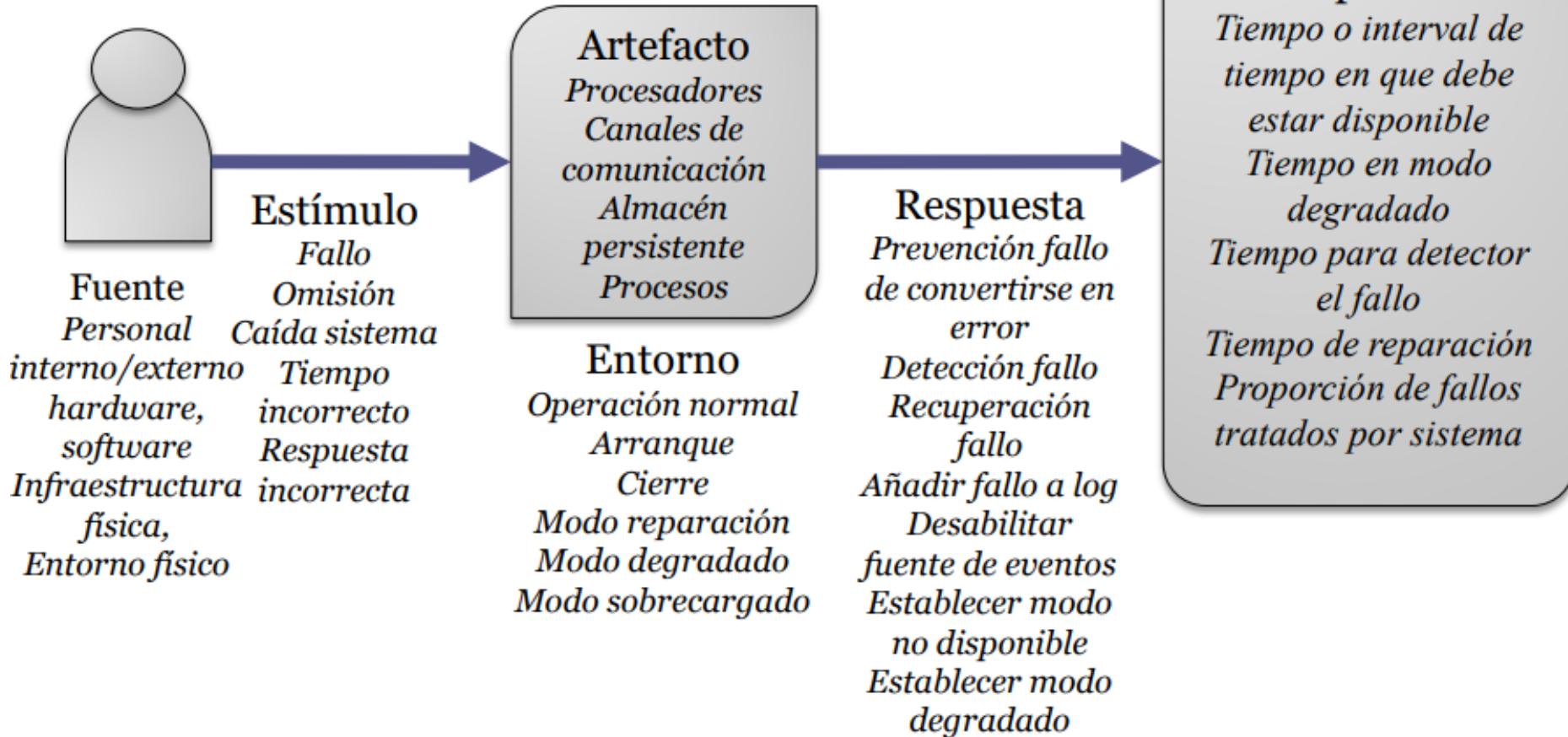
Tácticas

ESCENARIO DE CALIDAD: EJEMPLO

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



ESCENARIO DE CALIDAD: EJEMPLO DISPONIBILIDAD



DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD): ESCENARIOS (FRAMEWORKS) Y TÁCTICAS

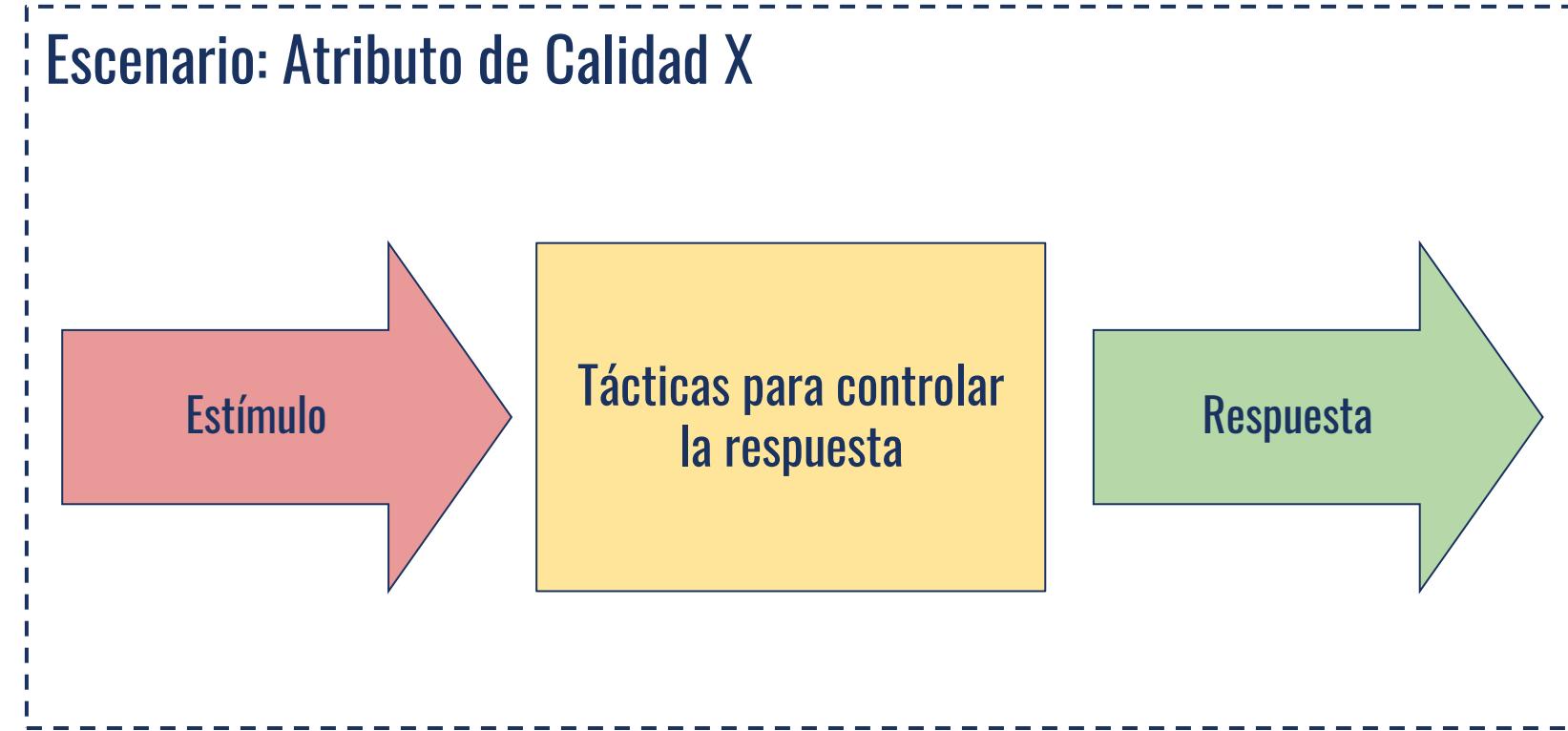
Framework de diseño orientado a atributos: Busca priorizar en el diseño, la solución e implementación los requerimientos, riesgos y atributos de calidad que fueron definidos para el proyecto. Plantea una estructura de escenarios y tácticas, en dónde cada escenario relaciona un atributos de calidad con distintas técnicas de implementación buscando mejorar la calidad.

1	Estímulo: cualquier hecho que afecte la calidad
2	Técnicas de respuesta: buscará controlar la respuesta al estímulo
3	Respuesta: respuesta esperada o caso de éxito al implementar la técnica

DISEÑO CENTRADO EN ATRIBUTOS DE CALIDAD (ADD): ESCENARIOS (FRAMEWORKS) Y TÁCTICAS

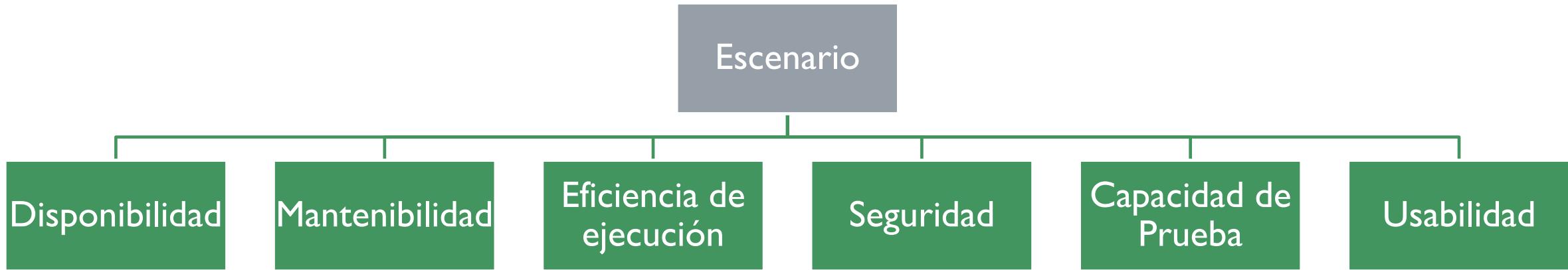
Andrés Armando Sánchez Martín

Escenario: Atributo de Calidad X



Algunos patrones y estilos ya implementan técnicas para el manejo de atributos

ESCENARIOS Y TÁCTICAS: CLASIFICACIÓN



Se clasifican según el atributo de calidad. Aunque no hay definición para todos los atributos de calidad definidos en la ISO/IEC 25000

ESCENARIOS Y TÁCTICAS: DISPONIBILIDAD

Escenario: Disponibilidad



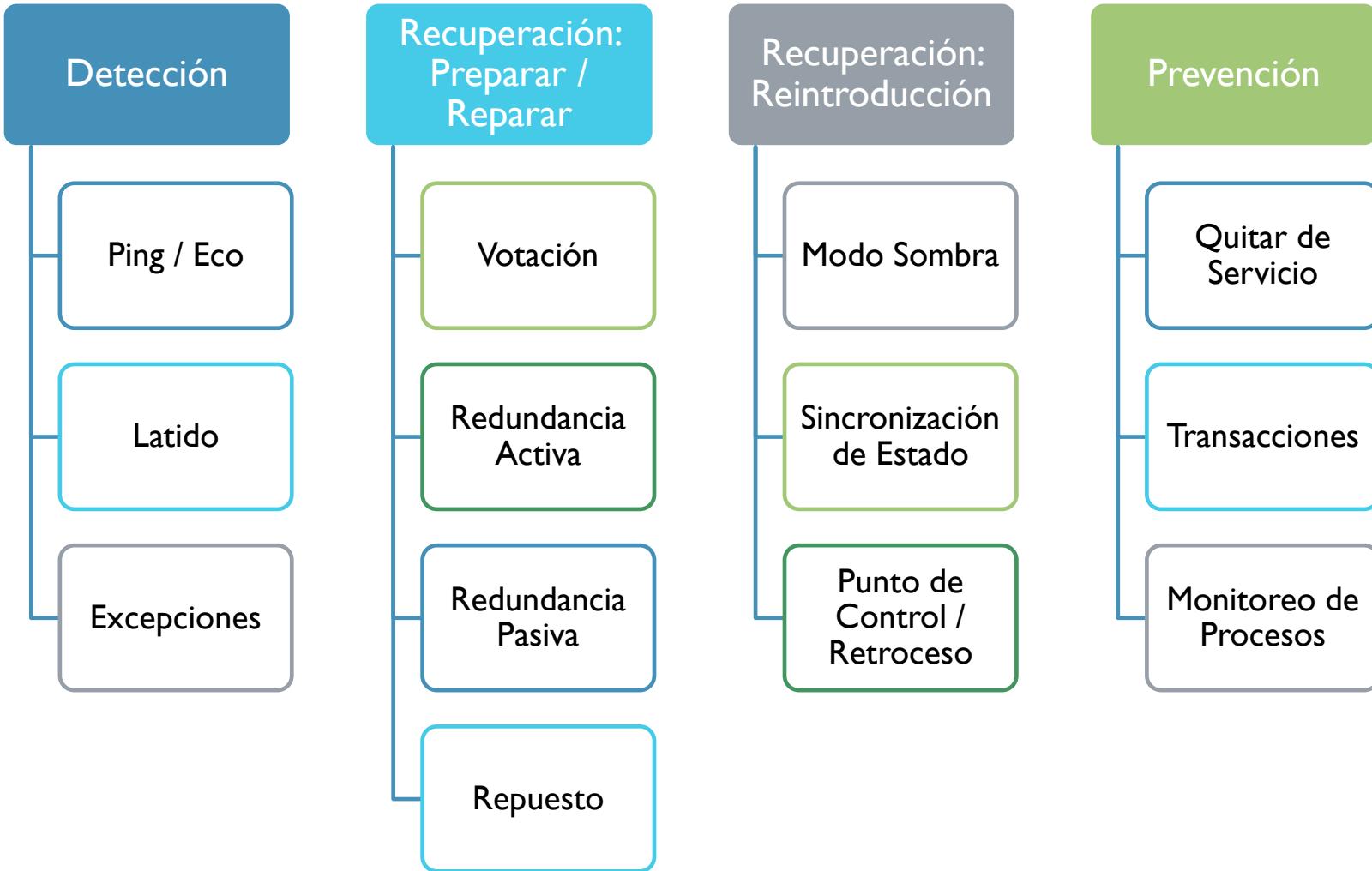
Falla

Tácticas para controlar
la disponibilidad

Falla ocultada o
sistema reparado

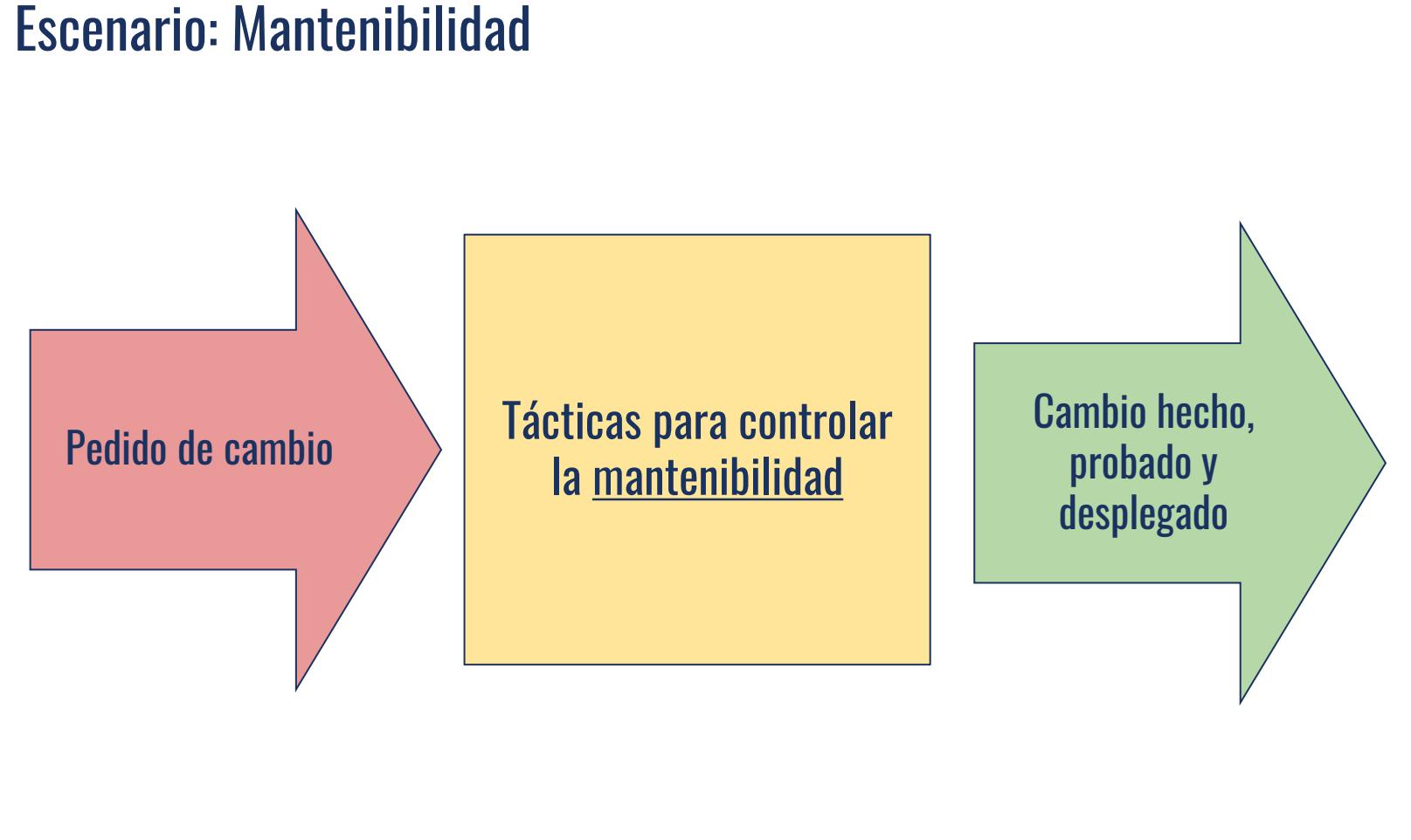
ESCENARIOS Y TÁCTICAS: DISPONIBILIDAD

Andrés Armando Sánchez Martín



ESCENARIOS Y TÁCTICAS: MANTENIBILIDAD

Escenario: Mantenibilidad

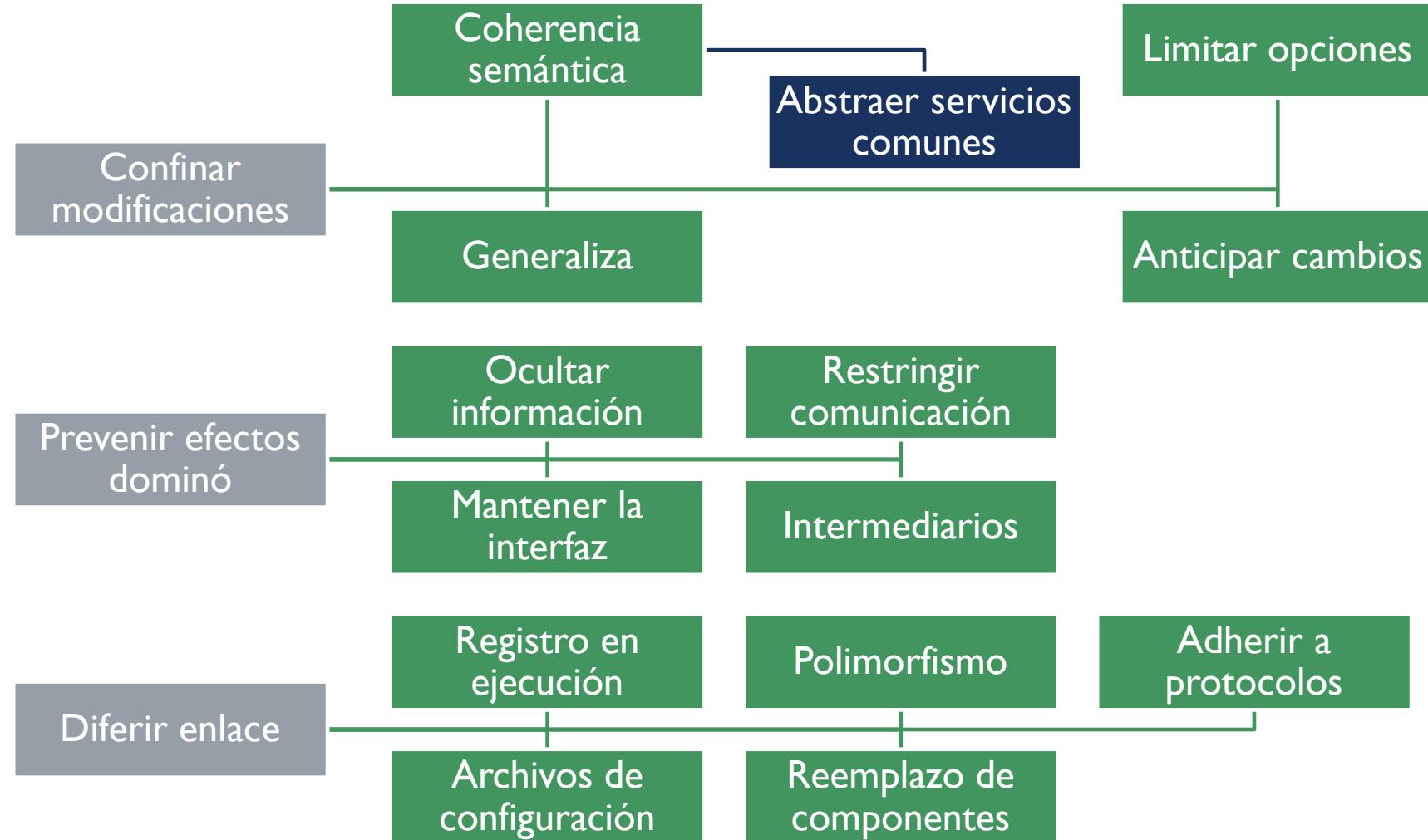


Pedido de cambio

Táticas para controlar
la mantenibilidad

Cambio hecho,
probado y
desplegado

ESCENARIOS Y TÁCTICAS: MANTENIBILIDAD

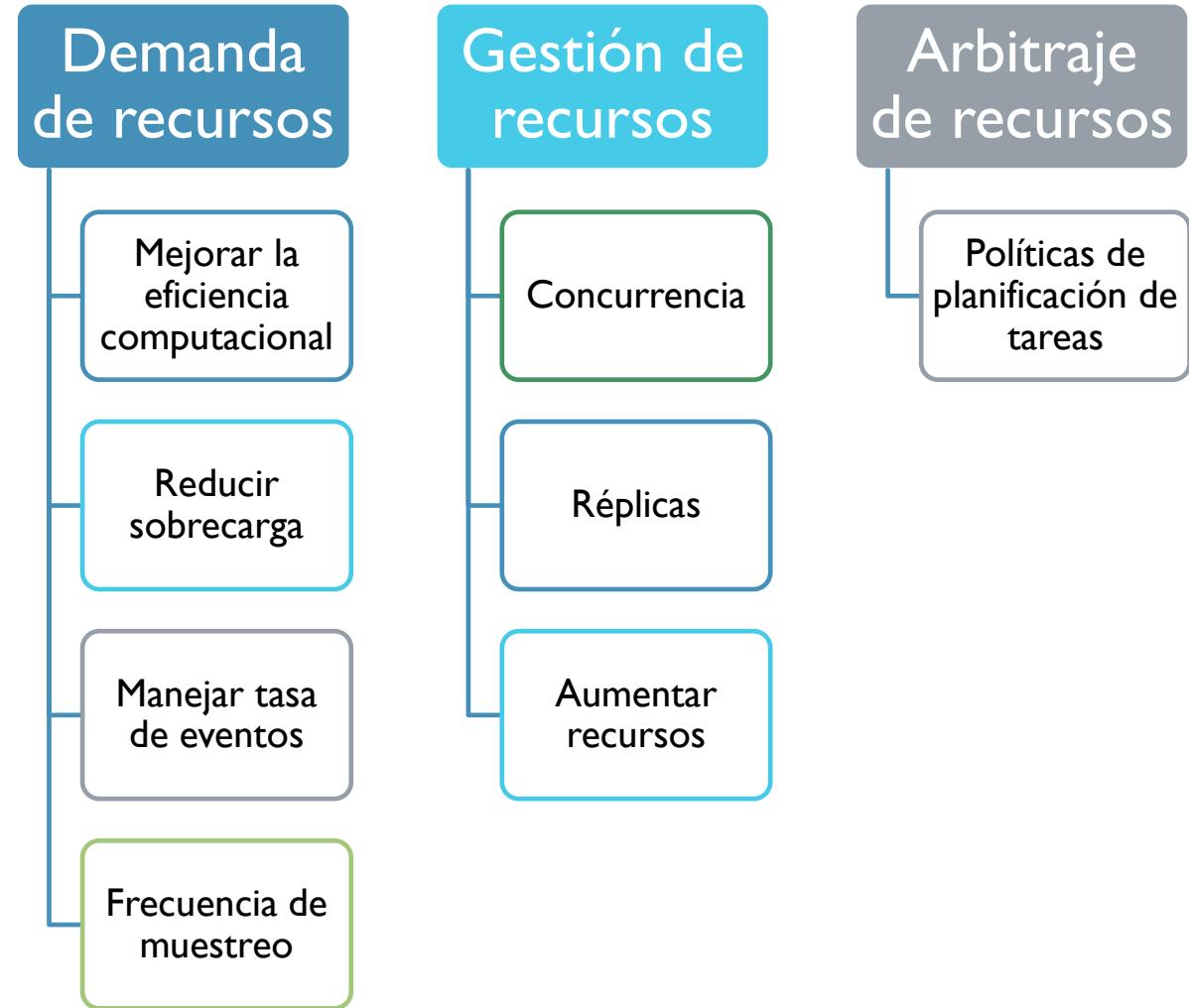


ESCENARIOS Y TÁCTICAS: EFICIENCIA DE EJECUCIÓN

Escenario: Eficiencia de ejecución

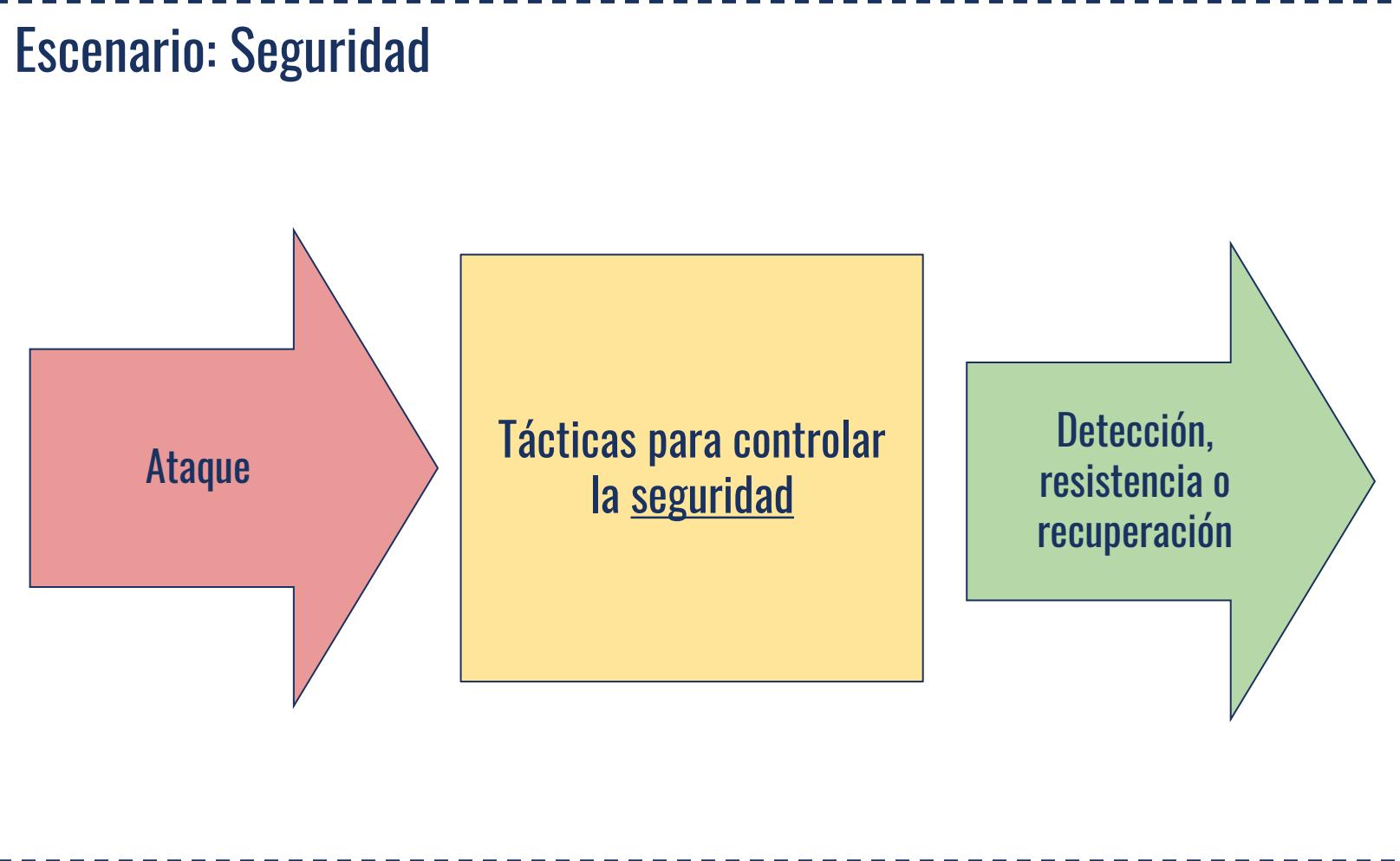


ESCENARIOS Y TÁCTICAS: EFICIENCIA DE EJECUCIÓN



ESCENARIOS Y TÁCTICAS: SEGURIDAD

Escenario: Seguridad

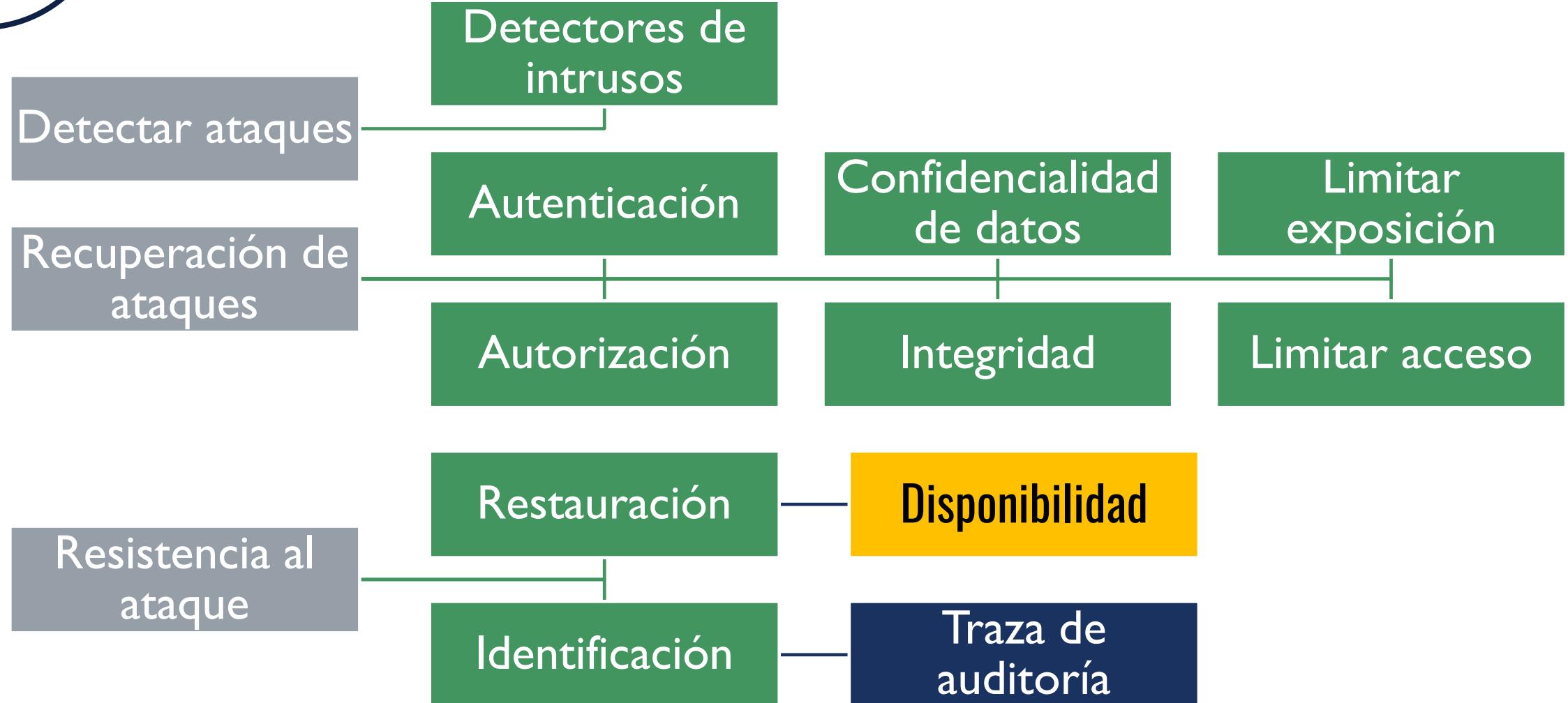


Ataque

Tácticas para controlar
la seguridad

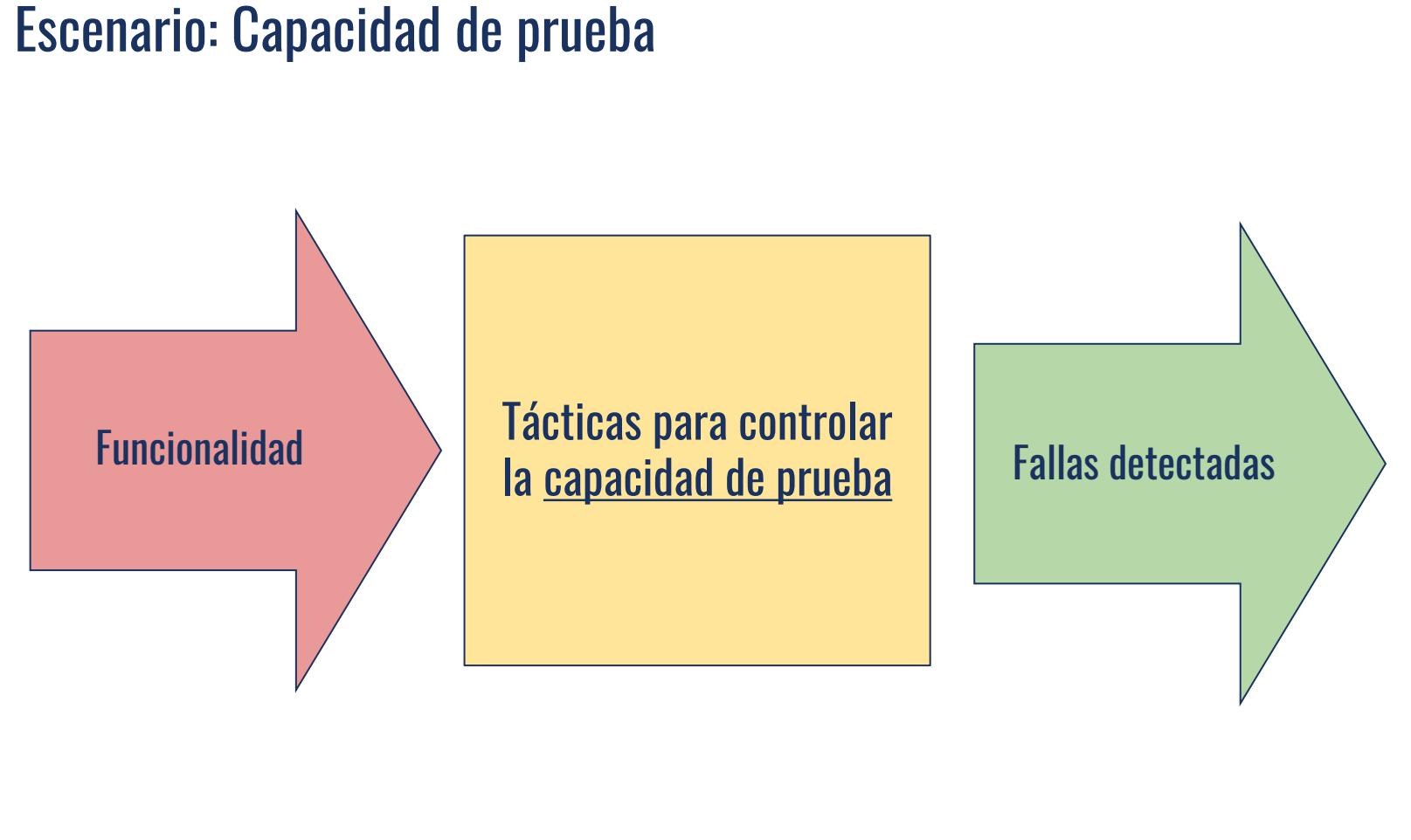
Detección,
resistencia o
recuperación

ESCENARIOS Y TÁCTICAS: SEGURIDAD



ESCENARIOS Y TÁCTICAS: CAPACIDAD DE PRUEBA

Escenario: Capacidad de prueba



Funcionalidad

Tácticas para controlar
la capacidad de prueba

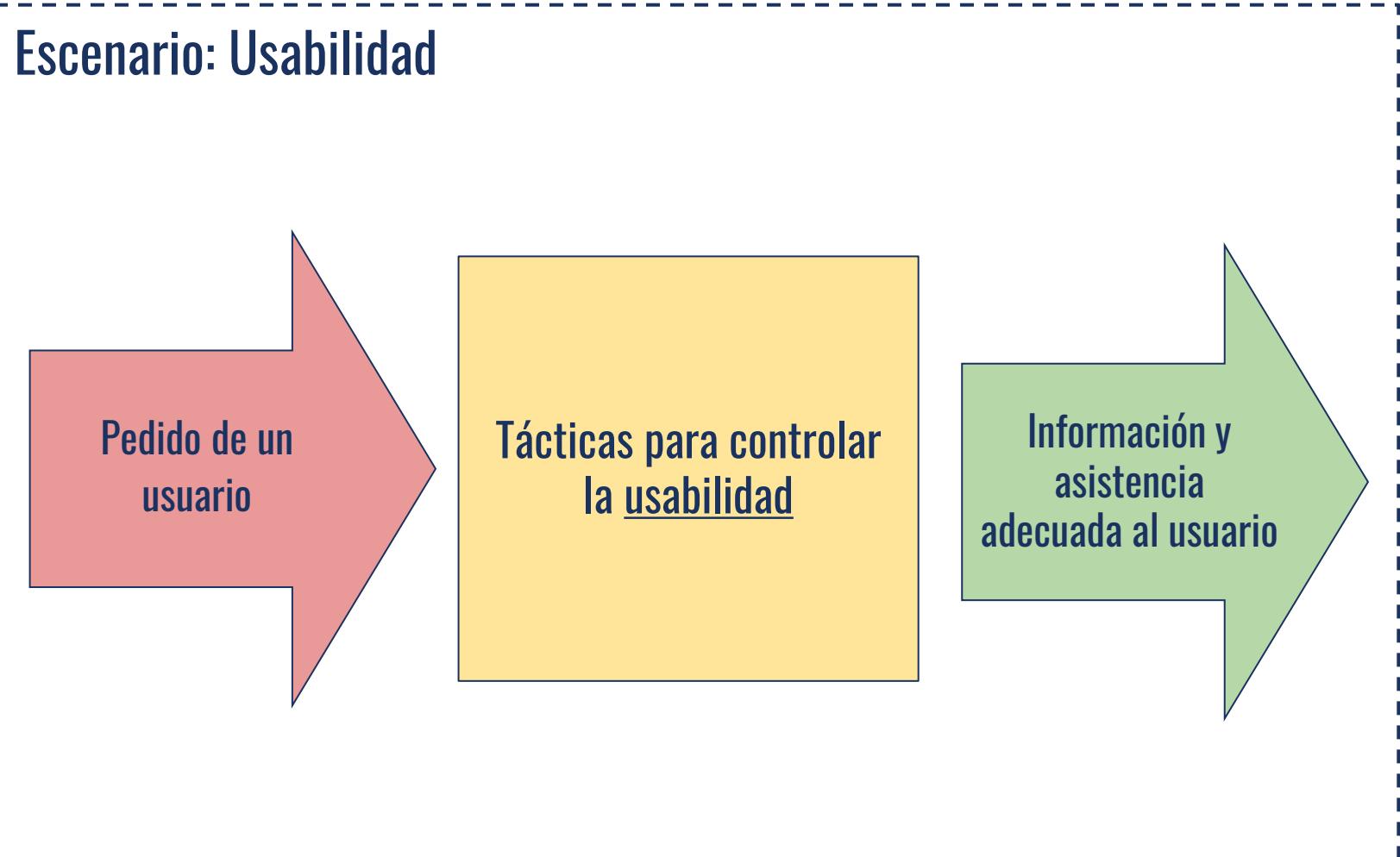
Fallas detectadas

ESCENARIOS Y TÁCTICAS: CAPACIDAD DE PRUEBA



ESCENARIOS Y TÁCTICAS: USABILIDAD

Escenario: Usabilidad

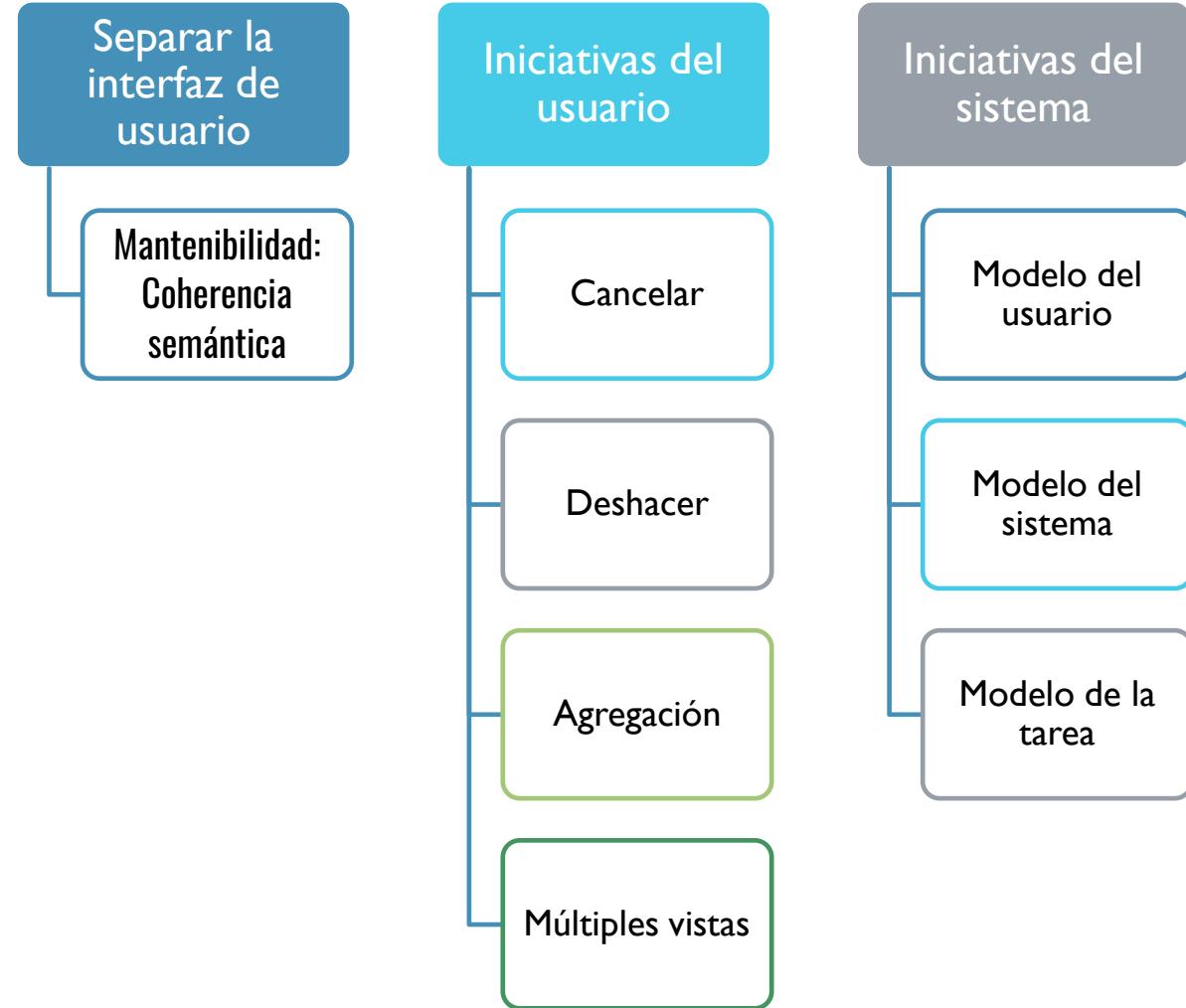


Pedido de un
usuario

Tácticas para controlar
la usabilidad

Información y
asistencia
adecuada al usuario

ESCENARIOS Y TÁCTICAS: USABILIDAD



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Cómo se valida una Arquitectura?
- ¿Qué es ATAM?
- ¿Qué es Prototipar?
- ¿Qué es MVP?





(1306)

ARQUITECTURA DE SOFTWARE

7.1 Proceso de Arquitectura de Software 3

Agenda

- Proceso de Arquitectura:
- Requerimientos
- Diseño
- Validación
 - ATAM
 - Escenarios
 - Prototipado



PROCESO DE ARQUITECTURA DE SOFTWARE

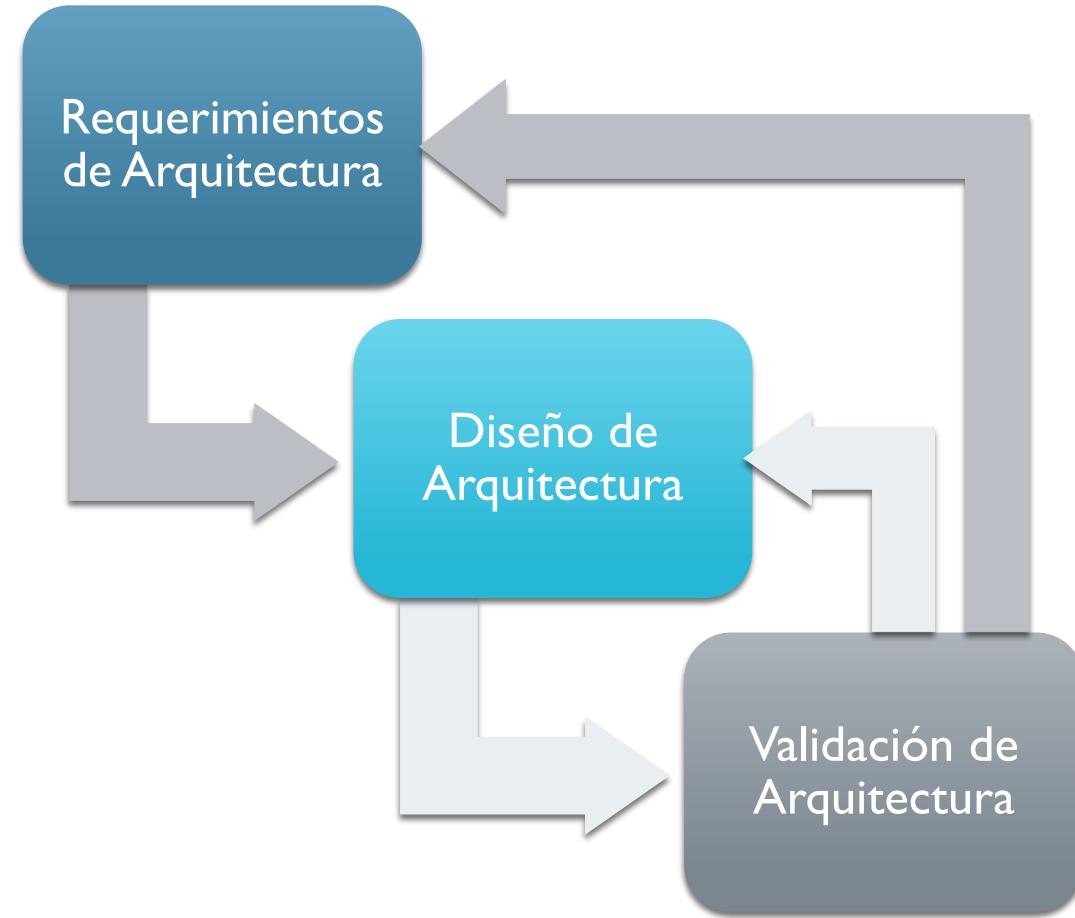
Identificar y especificar los requerimientos guía. Éstos incluyen a los atributos de calidad, los requerimientos funcionales primarios, los riegos y las restricciones del sistema. La arquitectura se puede diseñar alrededor de estos.

Diseñar la arquitectura e, idealmente, implementar una “arquitectura ejecutable” que permita materializar el diseño de la arquitectura.

Documentar los aspectos fundamentales del diseño, teniendo especial cuidado en capturar las decisiones (acuerdos) de diseño, para comunicarlas al equipo y que sirvan de guía.

Realizar una evaluación poco después de que se ha terminado el diseño y antes de proceder a la construcción del sistema.

PROCESO DE ARQUITECTURA DE SOFTWARE



REQUERIMIENTOS DE ARQUITECTURA

Funcionales

- Selección de Requerimientos Significativos para la arquitectura
- Casos de uso de Arquitectura

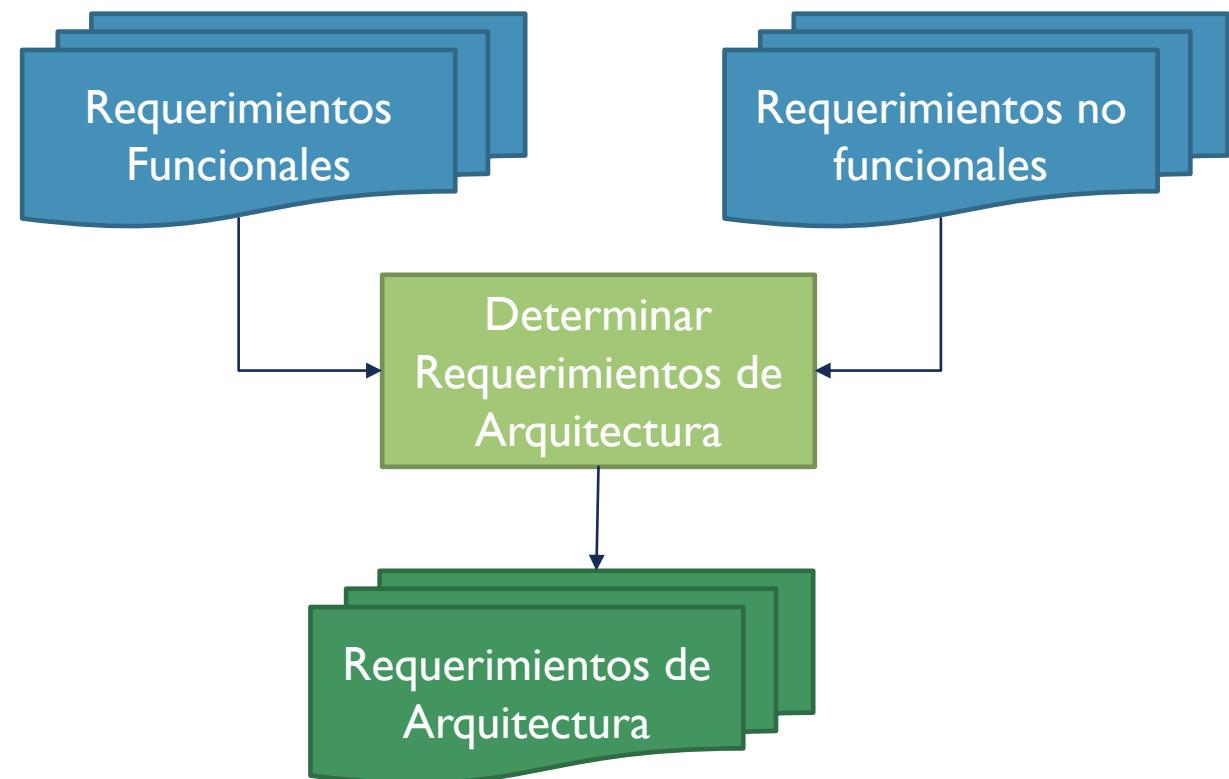
No funcionales

- Usabilidad
- Fiabilidad
- Rendimiento
- Compatibilidad
- Modificabilidad
- Seguridad
- Etc

Restricciones

- Negocio
- Desarrollo
- Físico
- Proyecto
- Arquitectura

REQUERIMIENTOS DE ARQUITECTURA



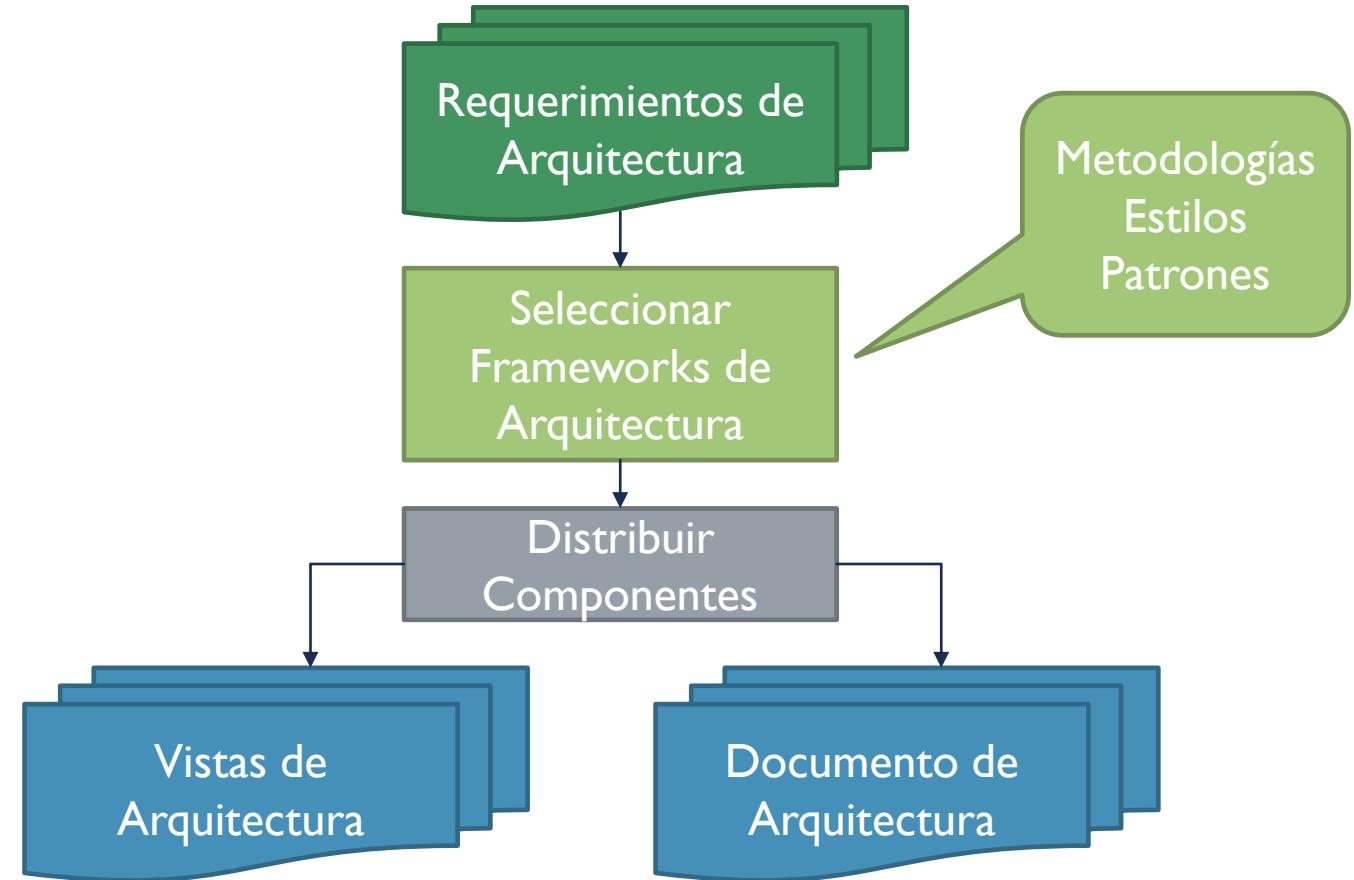
PRIORIZACIÓN DE REQUERIMIENTOS

- Todos los requisitos no son iguales
 - Alto: la aplicación debe soportar este requisito.
 - Medio: este requisito deberá cumplirse en algún momento
 - Bajo: esto es parte de la lista de deseos de requisitos.
- Difícil frente a los conflictos, por ejemplo:
 - Reutilización de componentes en la solución frente a rápido tiempo de comercialización. Hacer componentes generalizados y reutilizables siempre requiere más tiempo y esfuerzo.
 - Gasto mínimo en productos COTS versus esfuerzo/costo de desarrollo reducido. Los productos COTS significan que tiene que desarrollar menos código, pero cuestan dinero.
- Es diseño: ¡no pretende ser fácil!

COTS: Commercial off-the-shelf

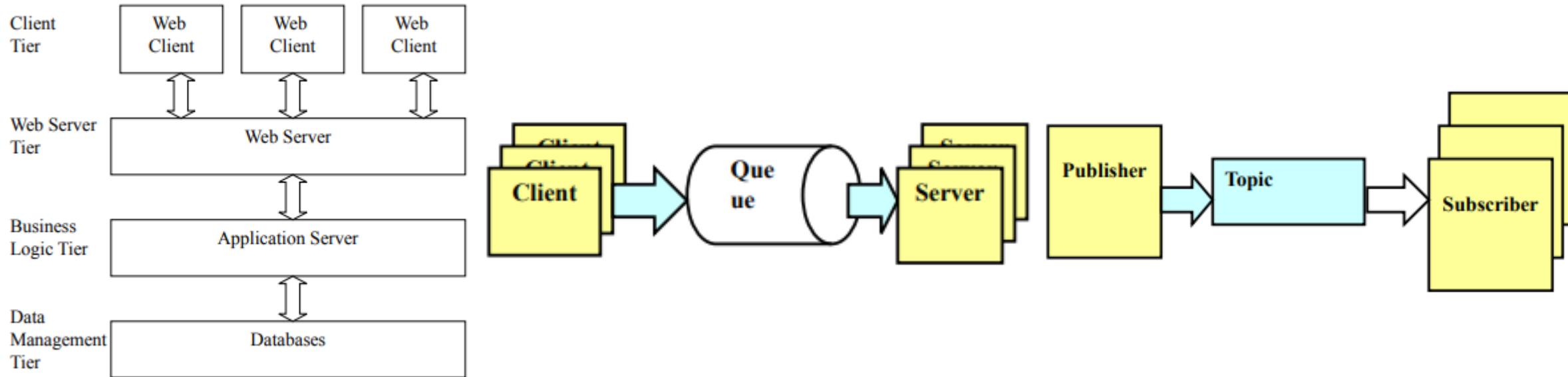
DISEÑO DE ARQUITECTURA

- Los pasos de diseño son iterativos
- La identificación de riesgos es crucial en el diseño.



FRAMEWORKS

	Disponibilidad	Manejo de fallas	Modificabilidad	Rendimiento	Escalabilidad
Capas	X	X	X	X	X
Mensajes	X	X	X	X	X
Publicador Suscriptor	X	X	X	X	X



DISTRIBUIR COMPONENTES

- Necesitar:
 - Identifique los principales componentes de la aplicación y cómo se conectan al marco.
 - Identifique la interfaz o los servicios que admite cada componente.
 - Identifique las responsabilidades del componente, indicando en qué se puede confiar cuando recibe una solicitud.
 - Identificar dependencias entre componentes.
 - Identificar particiones en la arquitectura que son candidatas para la distribución a través de servidores en una red y desarrollo independiente
 - Minimice las dependencias entre los componentes. Esfuércese por una solución débilmente acoplada en la que los cambios en un componente no se propaguen a través de la arquitectura, propagándose a través de
- muchos componentes.
 - Recuerde, cada vez que cambie algo, debe volver a probarlo.
 - Diseñe componentes que encapsulen un conjunto de responsabilidades altamente "cohesivo". La cohesión es una medida de qué tan bien encajan las partes de un componente.
 - Aíslle las dependencias del middleware y cualquier tecnología de infraestructura COTS.
 - Utilice la descomposición para estructurar los componentes jerárquicamente.
 - Minimice las llamadas entre componentes, ya que pueden resultar costosas si los componentes están distribuidos.

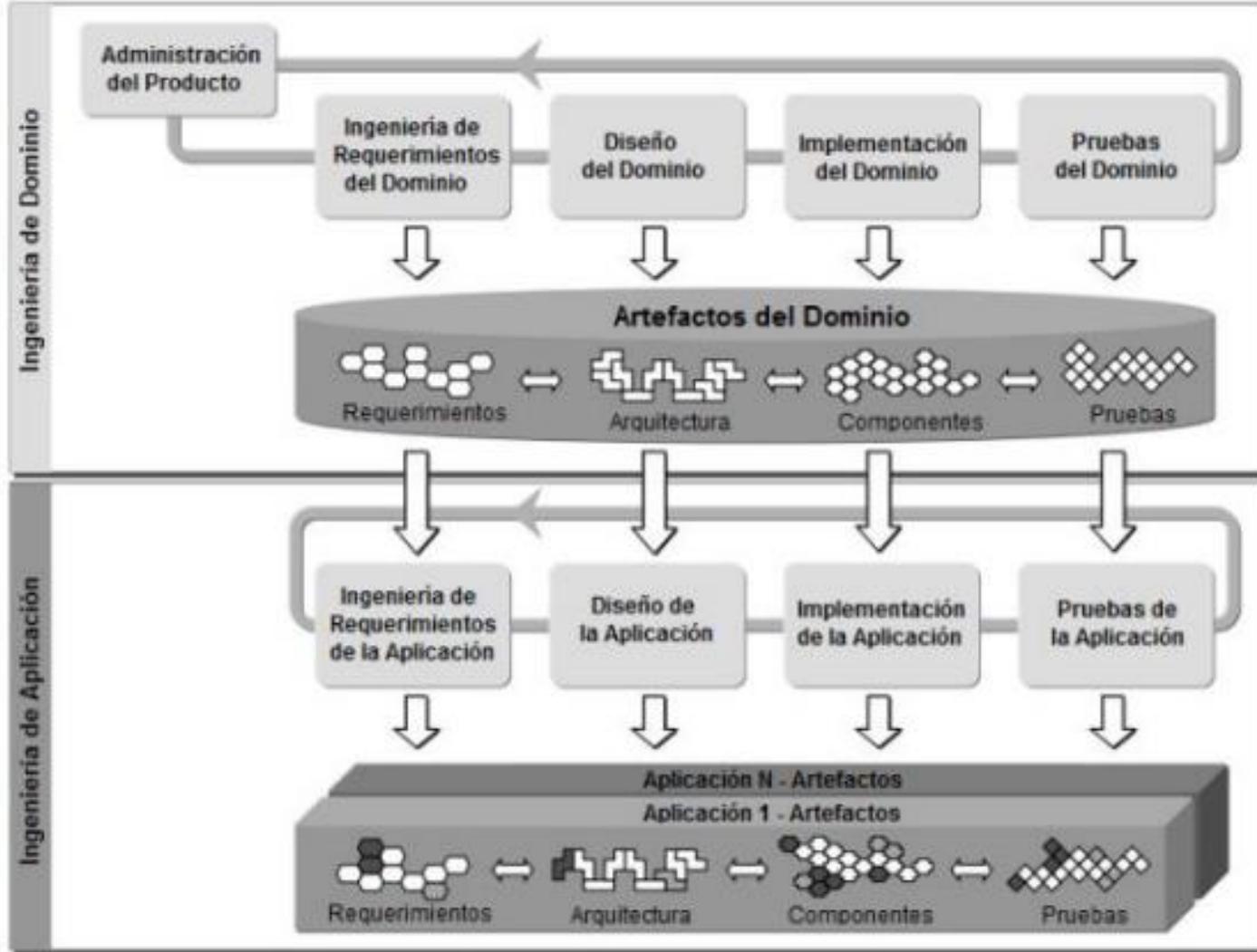
VALIDACIÓN DE ARQUITECTURA

- El Objetivo de la fase de validación es aumentar la confianza del equipo de diseño donde la arquitectura se ajuste para el propósito.
- La validación se ha de conseguir dentro de las restricciones del proyecto de tiempo y presupuesto.
 - El truco está en ser lo más riguroso y eficiente posible.
- Validar un diseño de la arquitectura plantea retos difíciles como:
 - Aquel diseño que no se puede ejecutar o probado.
 - Consta de componentes nuevos y COTS que deben integrarse.

VALIDACIÓN DE ARQUITECTURA

- Dos Técnicas Maestras:
 - Pruebas manuales de la arquitectura utilizando escenarios de prueba.
[ASAM, ATAM, ...]
 - Construcción de un prototipo que crea un sencillo arquetipo de la aplicación deseada.
- El objetivo de ambos es identificar posibles deficiencias en el diseño de manera que se pueden mejorar antes de que comience la ejecución.
- Más barato para arreglar antes de su construcción.

OBJETIVO DE LA VALIDACIÓN



ATAM: ARCHITECTURE TRADEOFF ANALYSIS METHOD

El Método de Análisis de Acuerdos de Arquitectura, es un método de evaluación de arquitectura de software desarrollado e impulsado por el Instituto de Ingeniería de Software, (Software Engineering Institute, SEI), este centra su actividad de evaluación en la interacción entre los diferentes atributos de calidad arquitectónica y basa sus evaluaciones sobre los escenarios desarrollados por los involucrados y un equipo de evaluación.

1^{ro} "sensibilidad" o "trade-off",

2^{do} marco para razonar sobre el sistema.

3^{ro} lista de cuestiones no abordadas o decisiones que no sean tomado en cuenta todavía

ATAM: ARCHITECTURE TRADEOFF ANALYSIS METHOD

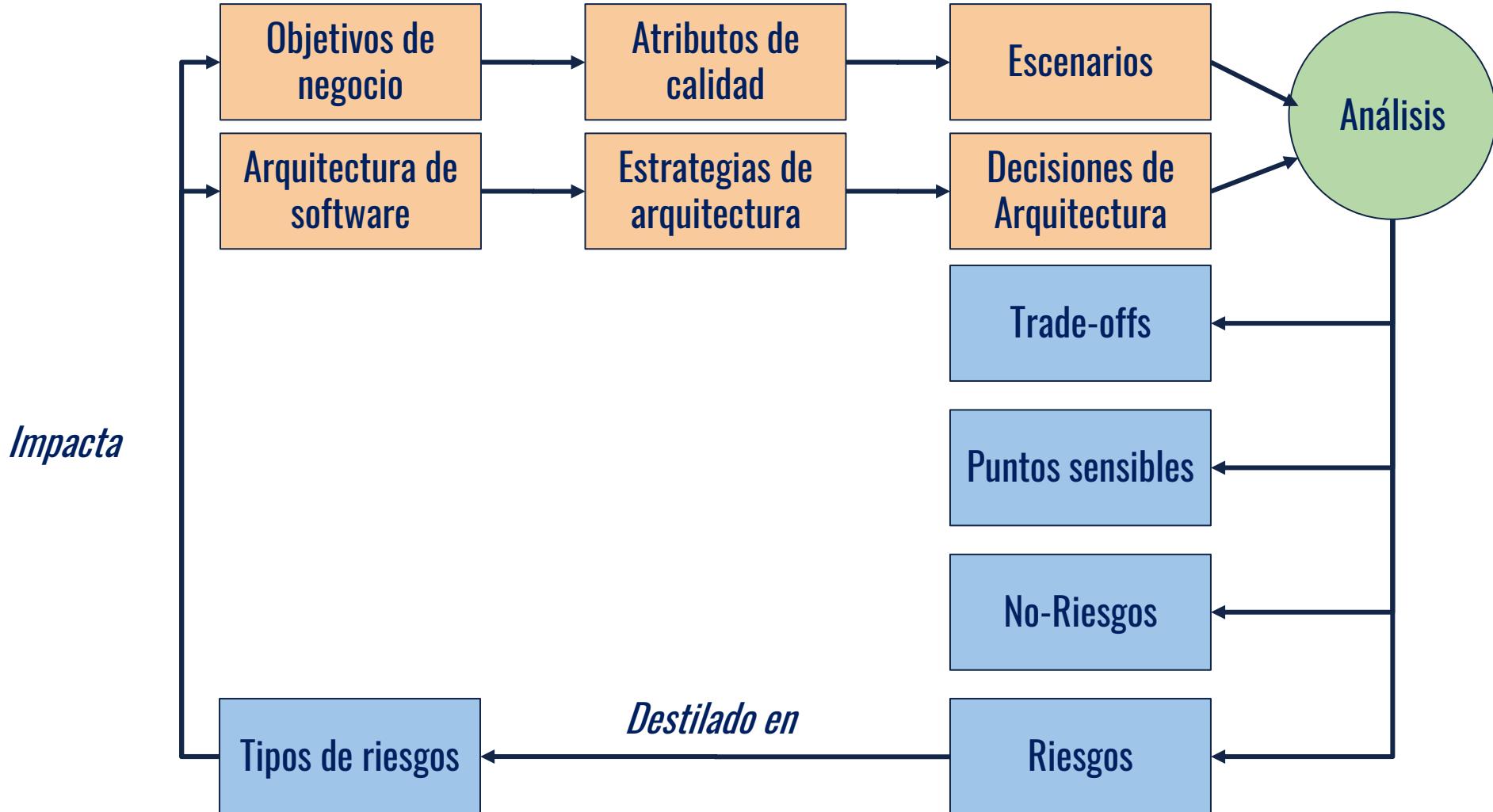
Descripción de las Vistas de Arquitectura y estilos

Recopilación y cartografía de los escenarios

Identificación de Riesgos / Sensibilidad / soluciones de compromiso



PROCESO ATAM



VALIDACIÓN DE ARQUITECTURA: ESCENARIOS

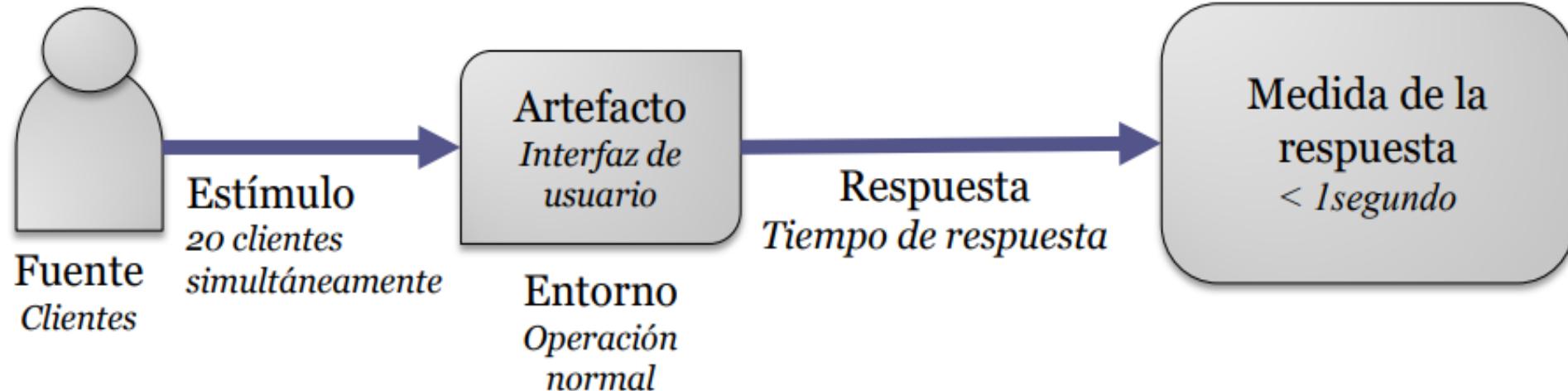
- Si la respuesta es deseable, un escenario se considera satisfecho para la arquitectura.
- Si la respuesta no es deseable, o difícil de cuantificar, entonces un defecto o por lo menos un área de riesgo se ha descubierto en la arquitectura.

Quality Attribute	Estímulo	Respuesta
Disponibilidad	La conexión de red de los consumidores de mensajes falla.	Los mensajes se almacenan en el servidor MOM hasta que la conexión se restablece. Los mensajes sólo se perderán si el servidor falla antes de que se restablezca.
Modificabilidad	Un nuevo conjunto de componentes de análisis de datos deben estar disponibles en la aplicación.	La aplicación necesita ser reconstruida con las nuevas librerías, y toda la configuración de los archivos debe ser actualizada en todos los escritorios para hacer los nuevos componentes visibles en la GUI.
Seguridad	No se reciben peticiones durante la sesión de un usuario durante 10 minutos.	El sistema toma esta sesión como potencialmente insegura e invalida las credenciales de seguridad asociadas a dicha sesión. El usuario debe ingresar nuevamente para acceder a la aplicación.
Modificabilidad	El proveedor del motor de transformación se quiebra.	Se debe contactar un nuevo proveedor. La capa de servicio abstracto que envuelve el motor de transformación debe ser implementada nuevamente. Los componentes del cliente no deben ser afectados ya que utilizarán el servicio abstracto.
Escalabilidad	La carga de solicitudes de usuarios simultáneos se duplica durante un periodo de tres semanas.	Se debe escalar la máquina servidor en dos clúster para así manejar las cargas de petición.

VALIDACIÓN DE ARQUITECTURA: ESCENARIOS

- Algún tipo de estímulo que tendrá un impacto en la arquitectura
- Trabajar en como la arquitectura responde a este estímulo

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

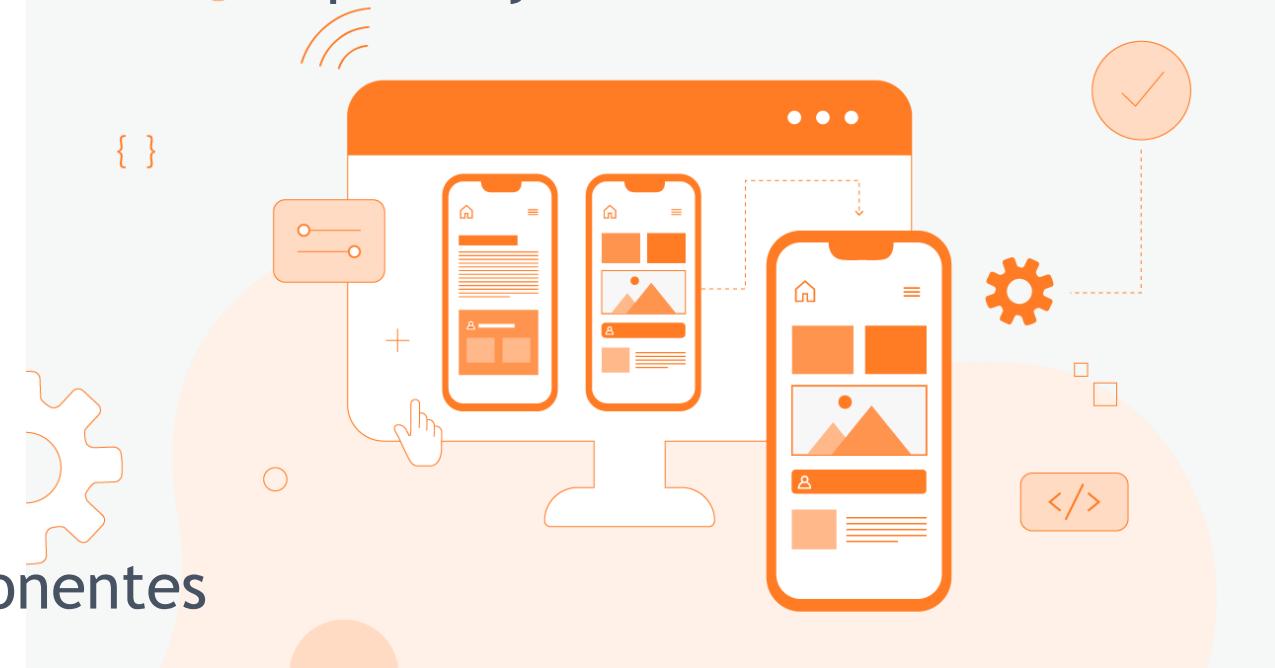


PROTOTIPADO

Los prototipos deben ser usados juiciosamente para ayudar a reducir los riesgos inherentes en un diseño.

La única forma de abordar:

- Desempeño
- Escalabilidad
- Fácil de integrar
- Capacidades de los mismos componentes



PROTOTIPADO

Necesitan ser cuidadosamente alcanzados y administrados.

- Idealmente toma un día o dos, una o a lo sumo dos semanas.
- Usualmente se desechan, así que manténgalos baratos
- No los deje adquirir una vida propia.



ESTRATEGIA DE PROTOTIPADO

- Construir un sistema mínimo requiere validar la arquitectura. Ej:
 - Una aplicación existente muestra que la cola y sistema de correo son capaces de soportar cinco mil mensajes en cinco minutos.
- Entonces:
 - Escriba un programa de prueba que llame al API sistema de validación de clientes cinco mil veces, y cuente cuanto tiempo tarda.
 - Escriba un programa de prueba que llame al API del sistema de ordenes de la tienda cinco mil veces, y cuente cuando tiempo tarda.

PROTOTIPO

- Los escenarios no pueden abordar todo:
 - "Un viernes en la tarde, las ordenes deben ser procesadas antes del cierre del negocio para asegurar la entrega para el lunes. Cinco mil ordenes llegan a través de varios canales (Web/centro de llamadas/compañeros de negocio) cinco minutos antes del cierre del negocio."
- Hay solo una manera - !Construir algo!
 - Prototipo Prueba de Concepto: ¿puede la arquitectura como diseño ser construida en una forma que satisfaga los requerimientos?
 - Prototipo Prueba de Tecnología: ¿la tecnología seleccionada (middleware, aplicaciones integradas, librerías, etc) implementa el comportamiento de la aplicación como se espera?

VALIDACIÓN DE ARQUITECTURA

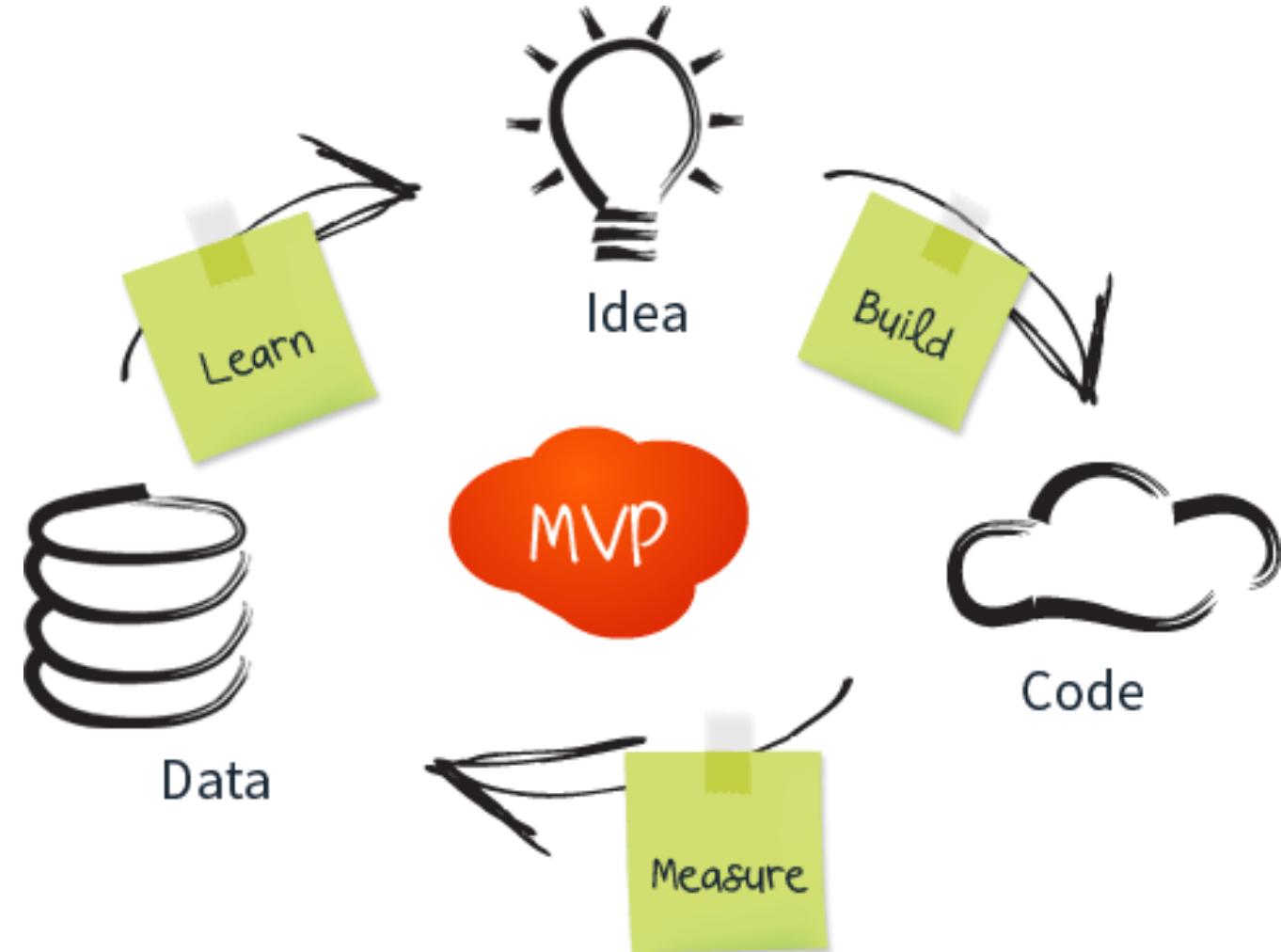
- Proceso de Validación
 - Asegurar si el sistema que se esta modelando con la arquitectura dada es el correcto.
 - Identificar personal clave y stakeholders internos en el proyecto:
 - Administrador de Proyecto
 - Desarrolladores
 - Testers
 - Tareas puntuales de validación
- Planear la Validación
 - Identificar personal clave que participa la validación.
 - Realizar cronogramas y reuniones programadas.
 - Distribuir materiales si hace falta antes de las reuniones
- Requerir del Arquitecto Líder y el Gestor del Proyecto.
- Revisar la Arquitectura
 - Presentar la arquitectura a los Stakeholders internos y resolver dudas.
 - Mostrar a grandes rasgos como la arquitectura está abordando a los requerimientos
 - Mostrar como hay trazabilidad entre los elementos de la arquitectura con respecto a los requerimientos
 - Hacer listas de chequeo para asegurar que todas las dudas y todos los requerimientos han sido abordados en la arquitectura presentada.

VALIDACIÓN DE ARQUITECTURA

- Documentar Hallazgos
 - Todo elemento que no se vea reflejado en la arquitectura o toda pregunta que quede sin responder debe quedar registrada.
 - Se deben considerar todos los comentarios encontrados de tal manera que se asegure que todo lo presentado en la arquitectura es completo y no-ambiguo.
- Evaluar Riesgos y hacer recomendaciones
 - Por cada Hallazgo, se debe evaluar su riesgo asociado.
 - Impacto potencial que genera en el proyecto
- Probabilidad de ocurrencia del riesgo
- Plantear una recomendación en forma de un plan de prevención o mitigación del riesgo, ya sea para evitar el riesgo o reducir el impacto que tiene sobre el proyecto o la probabilidad en la que ocurre
- Se deben registrar en los RAID log para asegurar que se han trazado y evaluar potenciales solicitudes de cambio.

MVP: PRODUCTO MÍNIMO VIABLE

- ¿Te gustaría comprobar tus hipótesis sobre el problema, la solución y el mercado de tu producto de forma económica?
- ¿Quieres obtener mejor información de tus usuarios?
- El MVP es una de las bases sobre las que se nace la metodología **Lean Startup**, y una de sus aportaciones más interesantes.



BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition.2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer.2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin.Wiley.2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley.2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en:
<http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



¿Preguntas?



PARA LA PRÓXIMA CLASE

Andrés Armando Sánchez Martín

- Entrega I del proyecto





(1306)

ARQUITECTURA DE SOFTWARE

7.2 Proceso de Arquitectura de Software 4



Pontificia Universidad
JAVERIANA
Bogotá

Agenda

Andrés Armando Sánchez Martín

- Proyecto



QUE SE ESPERA DEL PROYECTO

- Que identifiquen:

- Requerimientos de Arquitectura
- Atributos de calidad y escenarios
- Restricciones
- Riesgos

- Que hagan:

- Diseño de Alto Nivel (HLD)
- Diseño Lógico (LLD)
- Diseño de Detalle (Code/UML)

- Que apliquen

- Modelo C4 o modelo de 4 Vistas + I
- Aplique metodología para una arquitectura evolutiva
- Que validen la arquitectura

- Que entregue

- Modelos
- Documento
- Presentación





BIBLIOGRAFÍA / REFERENCIAS

1. Software Architecture in Practice, Third Edition. 2013. Len Bass, Paul Clements, Rick Kazman. disponible en Biblioteca General.
2. Essential Software Architecture. Ian Gorton. Springer. 2006. disponible en Biblioteca General.
3. The Art of Software Architecture. Stephen Albin. Wiley. 2003.
4. Documenting software architectures views and beyond 2nd ed., Clements, Paul. Addison-Wesley. 2011. disponible en Biblioteca General
5. The process of Software Architecting. Peter Eeles, Peter Cripps. Addison Wesley. 2010. disponible en Biblioteca General.
6. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Nick Rozanski y Eoin Woods. disponible en Biblioteca General.
7. Microsoft® Application Architecture Guide. 2nd Edition. David Hill. Patterns and practices. Disponible en: <http://www.microsoft.com/en-us/download/details.aspx?id=16236>
8. Beginning Java™ EE 7 Platform with GlassFish™ 3. From Novice to Professional Second Edition. Antonio Goncalves, Complementaria:
 1. Tutorial de JavaEE 8 <https://javaee.github.io/tutorial/>



¿Preguntas?



PARA LA PRÓXIMA CLASE

- ¿Qué es JavaEE / JakartaEE?





Arquitectura de Software

Arquitecturas empresariales con

J2EE/JavaEE/Jakarta

Java EE y Jakarta EE

- ▶ **in September 2017, Oracle decided to give away the rights for Java EE to the Eclipse Foundation (the language is still owned by Oracle).**
- ▶ Actually, the Eclipse Foundation legally *had* to rename Java EE. That's because Oracle has the rights over the “Java” brand. So, to choose the new name, the community voted and picked: **Jakarta EE**. In a certain way, it's still JEE.

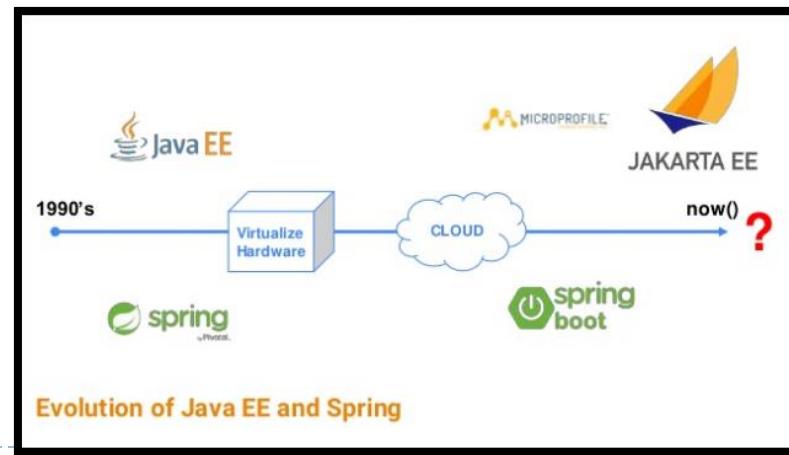
Version	Date
J2EE 1.2	December 1999
J2EE 1.3	September 2001
J2EE 1.4	November 2003
Java EE 5	May 2006
Java EE 6	December 2009
Java EE 7	April 2013
Java EE 8	August 2017
Jakarta EE	February 2018*

Java EE , Jakarta , Spring

- ▶ Java EE / Jakarta : ESTANDAR
- ▶ Spring: base Java, no sigue estrictamente los estándares
- ▶ Monilito



- ▶ Microservicios



Qué es JavaEE/J2EE

- ▶ **¿Que es Java Enterprise Edition?**
 - ▶ Es una arquitectura estándar
 - ▶ Provee un framework para construcción de componentes de presentación y negocios.
 - ▶ Simplifica el desarrollo de aplicaciones empresariales.
 - ▶ Provee servicios horizontales para seguridad, manejo de transacciones, ambiente multi-thread, manejo de recursos.
- ▶ **NO ES UN LENGUAJE DE PROGRAMACION.**

Beneficios de J2EE/JavaEE

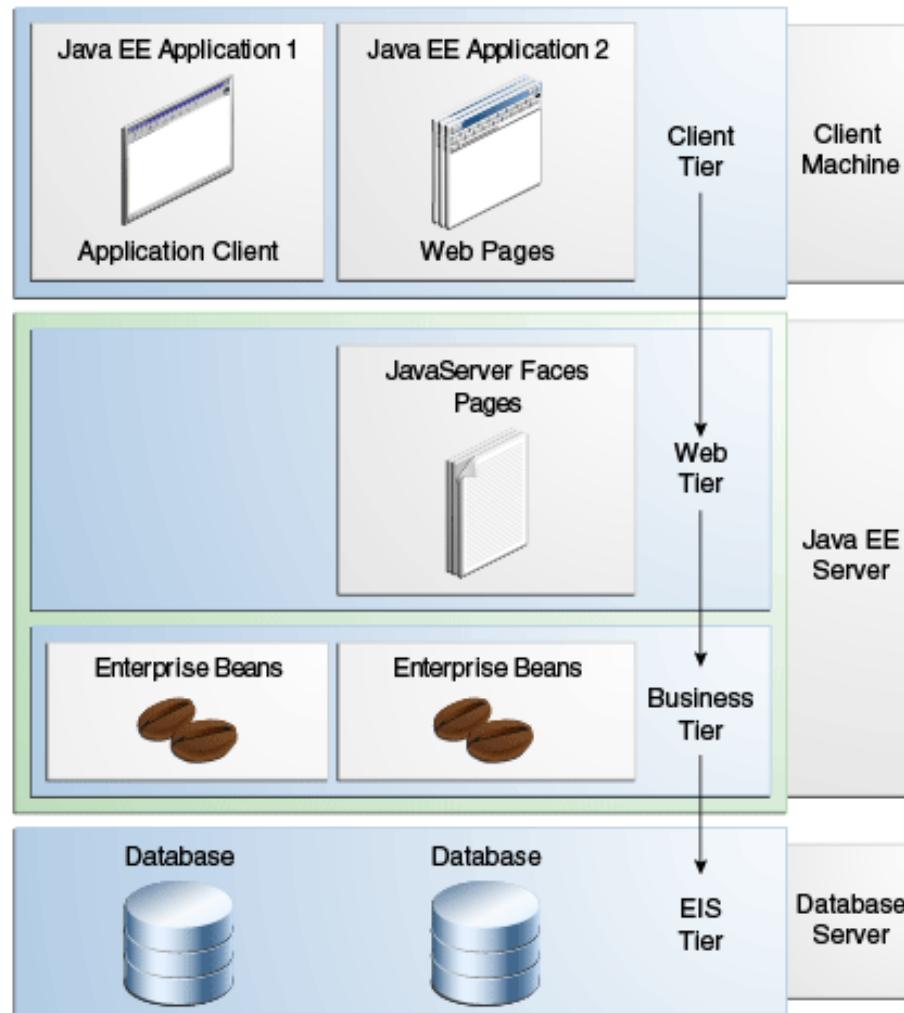
- ▶ **Arquitectura y desarrollo simplificado**
 - ▶ Modelo de desarrollo en componentes
 - ▶ Soportan división de tareas por roles
- ▶ **Libertad de escoger servidores, herramientas, y componentes**
- ▶ **Integración con sistemas existentes**
 - ▶ J2EE Connector Architecture permite interactuar con ERP, CRM, legacy, JDBC, JTA, JNDI, JMS, IDL, XML
- ▶ **Escalabilidad**
- ▶ **Modelo de seguridad flexible**
 - ▶ Mecanismo de seguridad basado en roles, no programático.

Modelo de Aplicaciones Java EE

- ▶ El modelo de aplicación Java EE define una arquitectura para la implementación de servicios como de aplicaciones de varios niveles que ofrecen:
 - ▶ la escalabilidad,
 - ▶ la accesibilidad
 - ▶ y capacidad de gestión necesaria para las aplicaciones de nivel empresarial



Aplicaciones MultiTier Distribuidas(1)

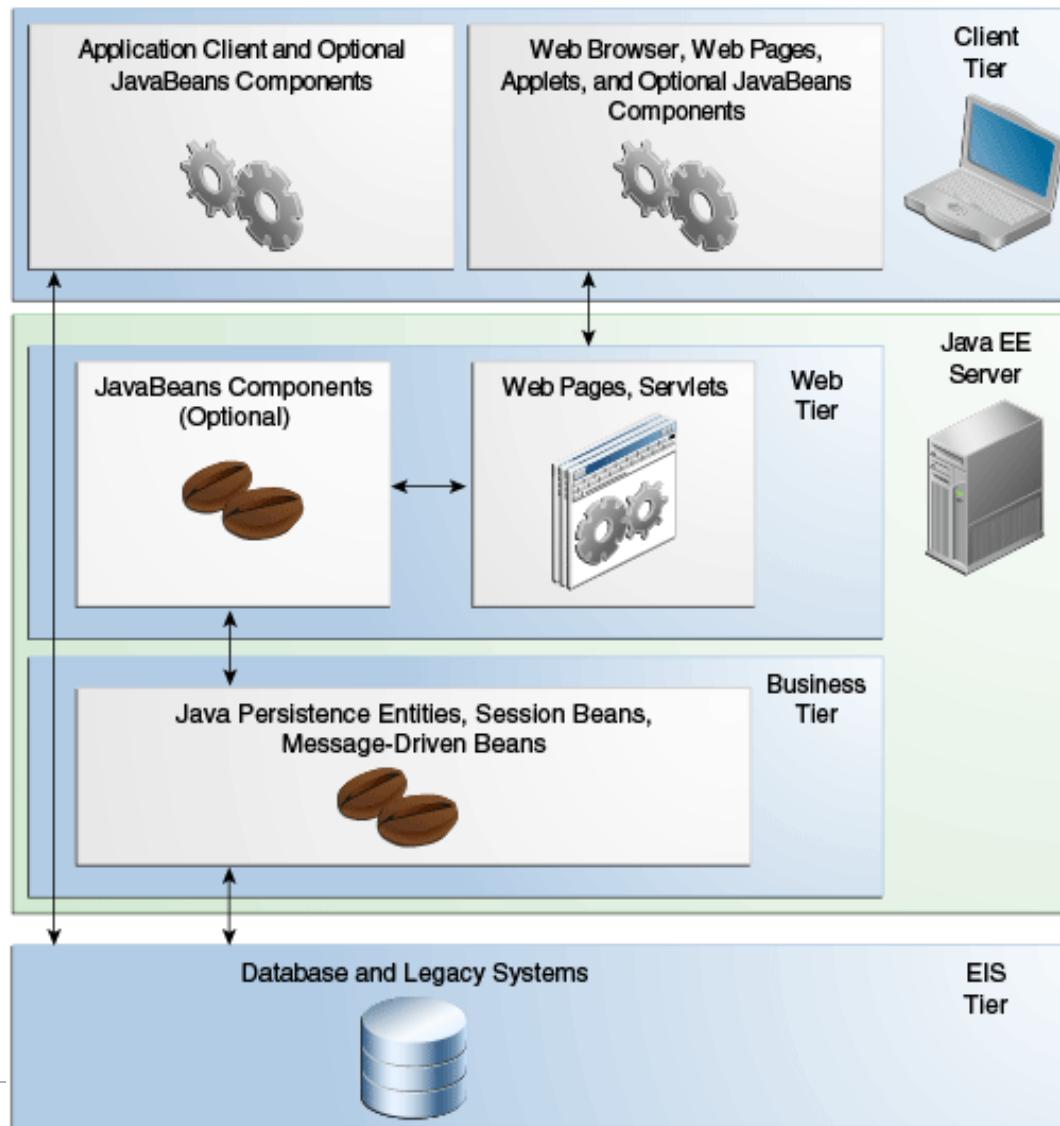


Java EE Tiers

Tiers	Descripción
Cliente	Máquina cliente o sistema que provee servicios de interacción con un humano para captura y despliegue de información.
Presentación	Provee servicios para crear dinámicamente las diferentes vistas que conforman la interfaz de usuario y envia mensajes al tier de negocios
Negocios	Provee las reglas que soportan las operaciones regulares de un negocio y ejecutan transacciones.
Integración	Servicios que permiten a un aplicativo abstraerse y de conectarse con recursos externos.
Recursos	Provee los servicios de almacenamiento de datos en un repositorio permanente (sistemas legados, bases de datos, directorios en línea, servidores mail y máquinas entre otros).

Aplicaciones MultiTier Distribuidas(2)

Interacciones entre tiers

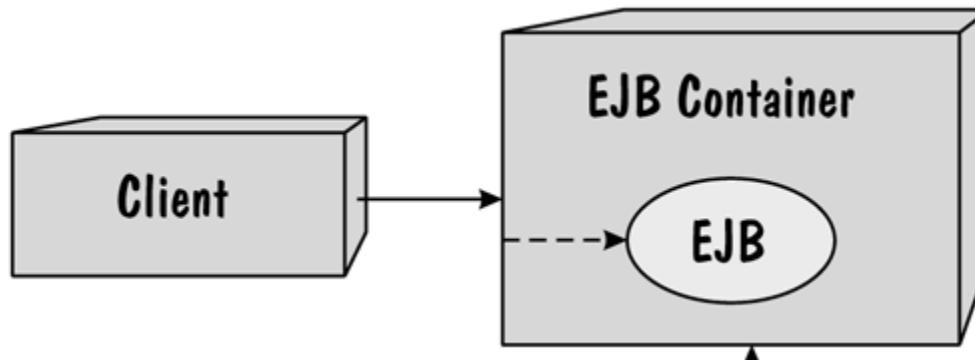


Containers

- ▶ Proveen ambiente de ejecución para los componentes
- ▶ Manejan el ciclo de vida de los componentes
- ▶ Provee servicios J2EE/JavaEE de manera transparente
 - ▶ Seguridad,
 - ▶ deployment,
 - ▶ threads,
 - ▶ transacciones,
 - ▶ persistencia

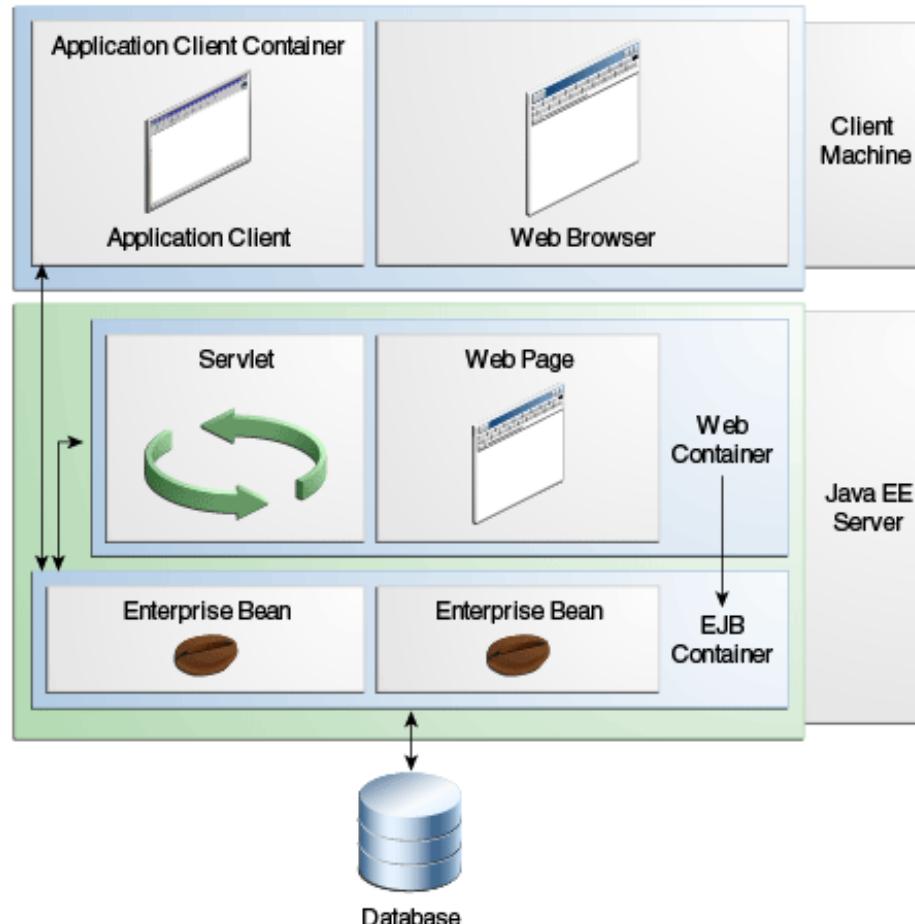
Containers

- ▶ Implementados como JVM
- ▶ Ejemplos:
 - ▶ Web Containers
 - ▶ Pueden automáticamente hacer balanceo de carga
 - ▶ Aloja los componentes Web
 - ▶ EJB Container
- ▶ El contenedor es transparente para el cliente



Containers(2)

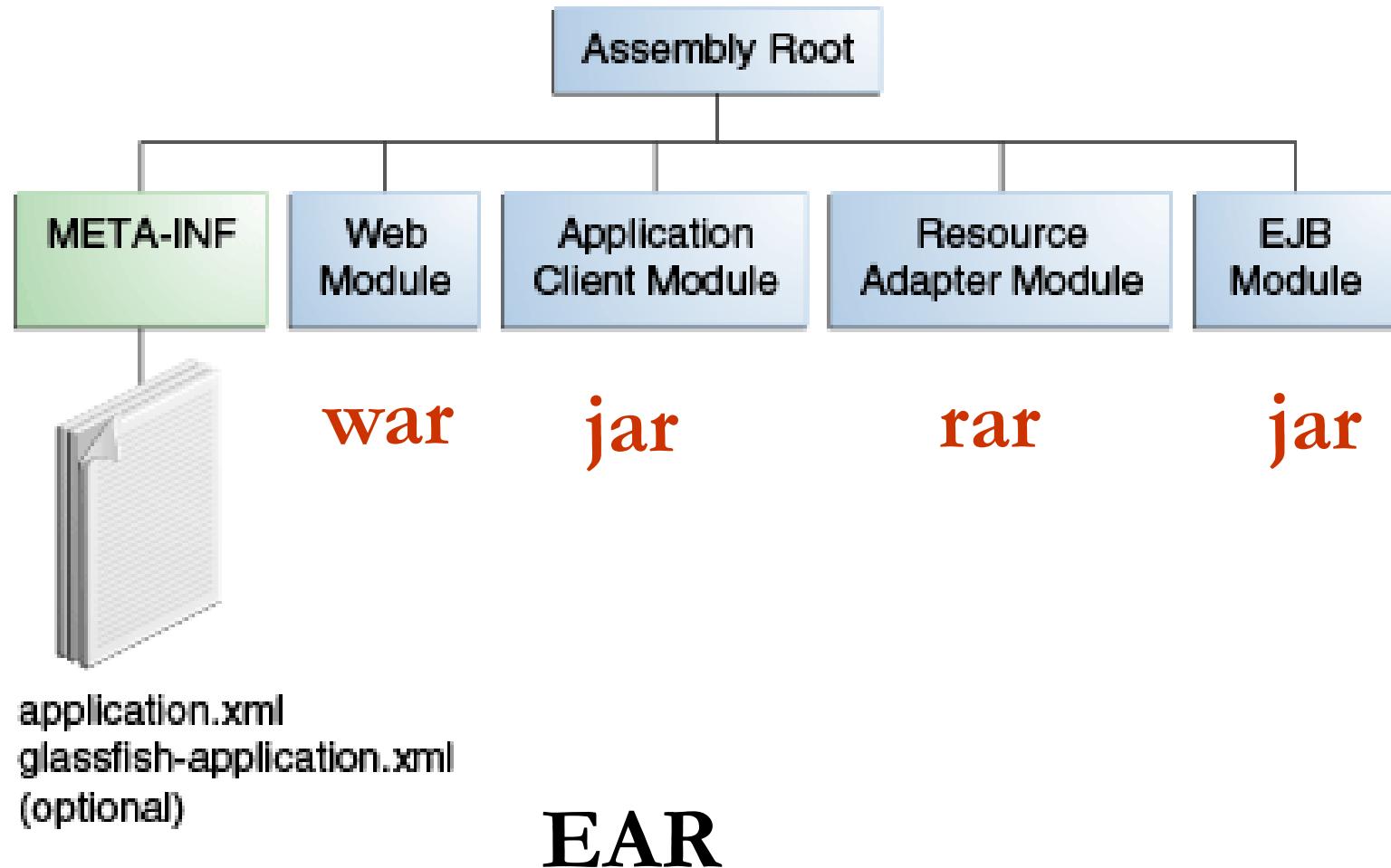
- ▶ Un servidor Java EE puede alojar varios contenedores



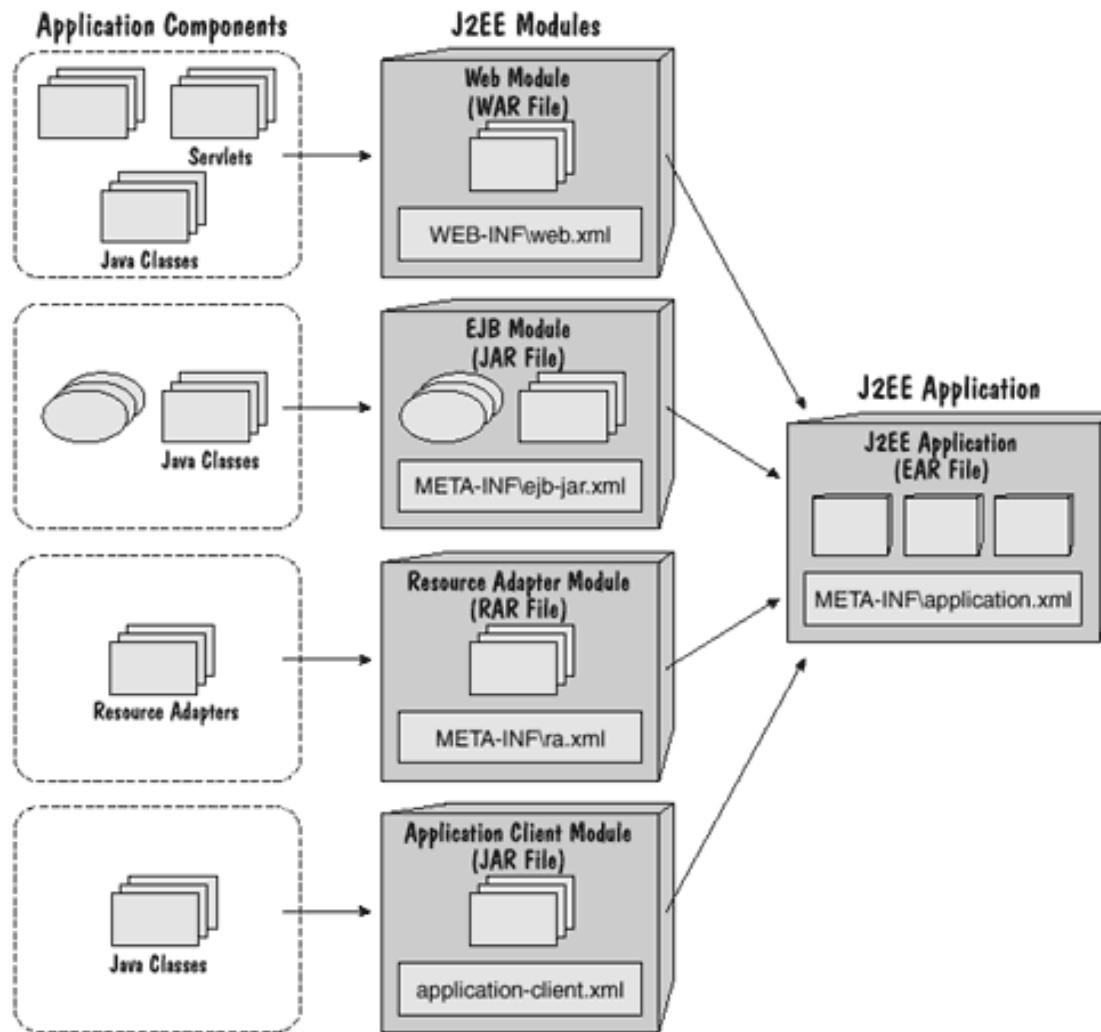
Soporte para Web Services

- ▶ Los servicios web son aplicaciones empresariales que usan estándares y protocolos abiertos basados en XML para intercambiar datos con sus clientes.
- ▶ JavaEE provee las API y herramientas XML para diseñar y desarrollar los WS
- ▶ Estándares soportados:
 - ▶ XML
 - ▶ SOAP
 - ▶ WSDL
 - ▶ UDDI

Empaquetar Aplicaciones(deploy)



Empaquetar Aplicaciones(deploy)





JNDI

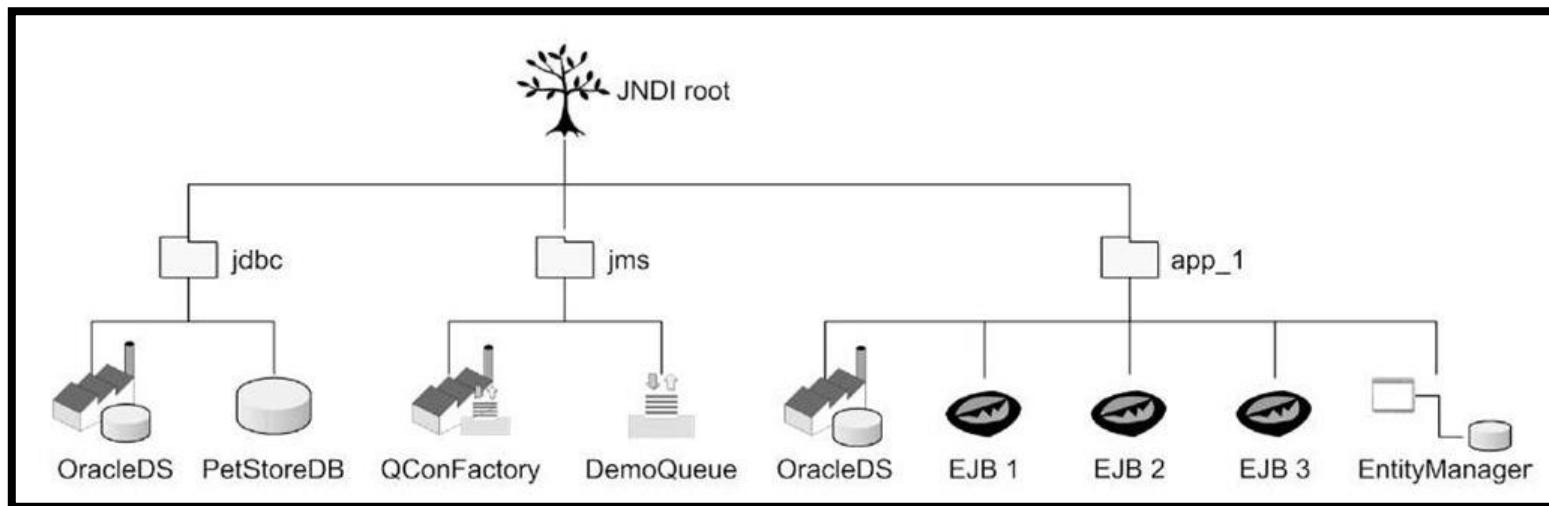
JNDI

- ▶ JNDI — Java Naming and Directory Interface API es un servicio de nomenclatura que permite a los componentes localizar otros componentes y recursos.



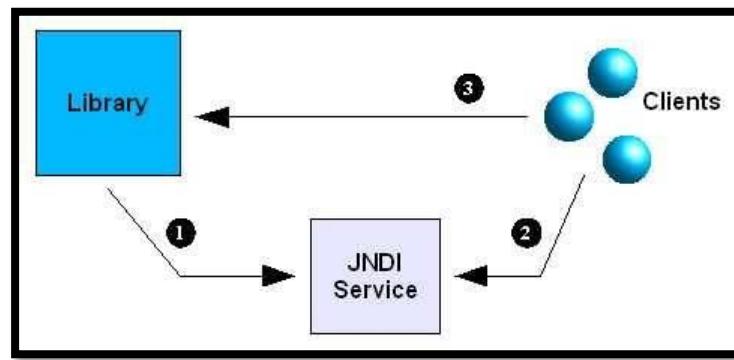
JNDI

- ▶ Evita ‘quemar’ URL o referencias de componentes
- ▶ Directorio de componentes



JNDI

- ▶ 1) Componente se registra (manual o automáticamente)
- ▶ 2) Clientes buscan por nombre
- ▶ 3) Los clientes acceden al componente



JNDI - JDBC

- ▶ JDBC — API de conectividad de base de datos Java, utilizada para ejecutar instrucciones SQL comunes y realizar otros objetivos comunes a las aplicaciones de base de datos.



JNDI - JDBC

- ▶ Registrar un pool de conexiones
- ▶ Instancias precreadas de objetos que permiten conexión eficiente a la base de datos

New JDBC Connection Pool (Step 2 of 2)

Identify the general settings for the connection pool. Datasource Classname or Driver Classname must be specified for the connection pool.

* Indicates required field

General Settings

Pool Name:	poolderbybooks
Resource Type:	javax.sql.DataSource
Database Driver Vendor:	Derby
Datasource Classname:	org.apache.derby.jdbc.ClientDataSource40
Driver Classname:	

Select or enter vendor-specific classname that implements the DataSource and/or XADatasource APIs

Select or enter vendor-specific classname that implements the java.sql.Driver interface.

Additional Properties (8)

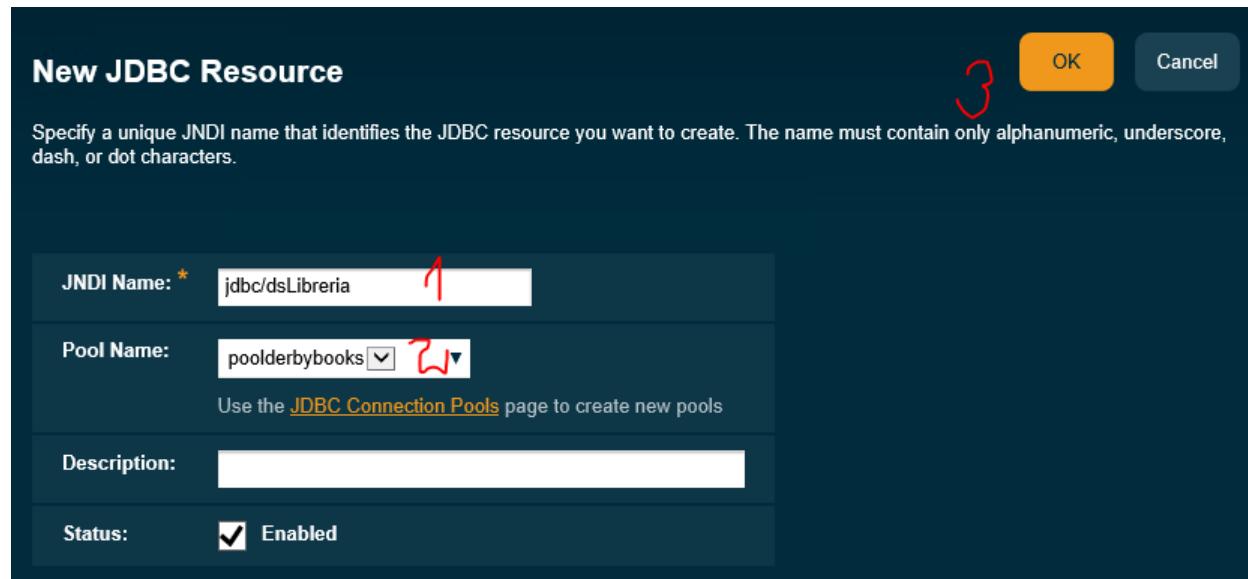
Select	Name	Value	Description
<input type="checkbox"/>	password	APP	
<input type="checkbox"/>	databaseName	sample	
<input type="checkbox"/>	roleName		
<input type="checkbox"/>	serverName	localhost	
<input type="checkbox"/>	datasourceName		
<input type="checkbox"/>	user	APP	
<input type="checkbox"/>	networkProtocol		
<input type="checkbox"/>	portNumber	1527	

Previous **Finish** Cancel

Bye

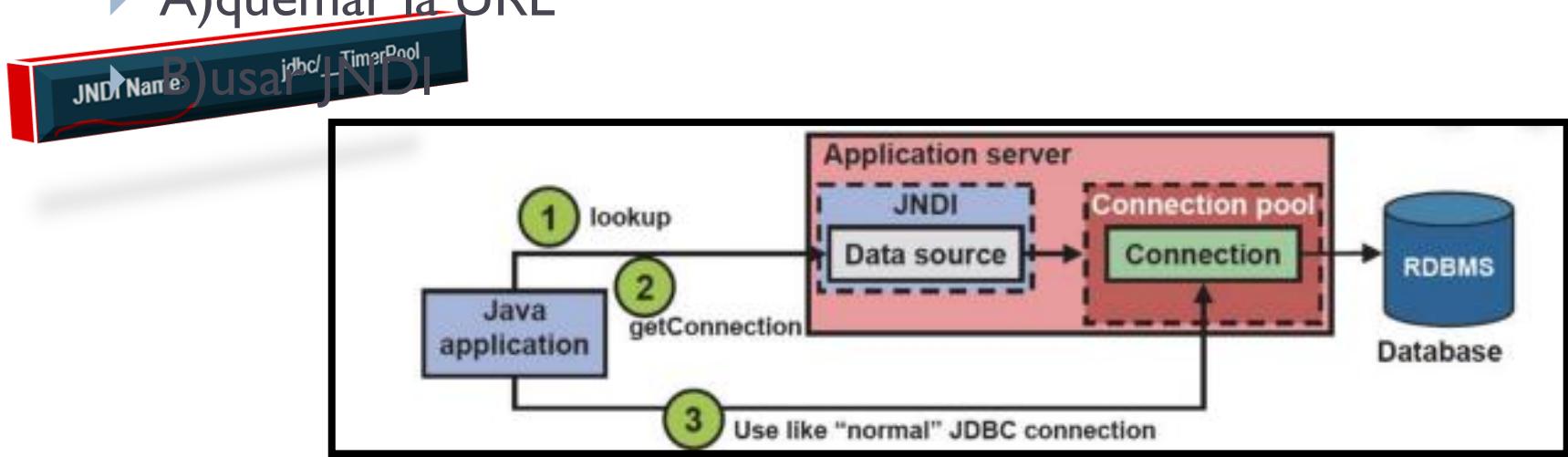
JNDI - JDBC

- ▶ Registrar un DataSource
- ▶ Tiene el jndi_name



JNDI - JDBC

- ▶ En este ejemplo el componente ‘Java Application’ quiere usar la base de datos
- ▶ A) quemar la URL



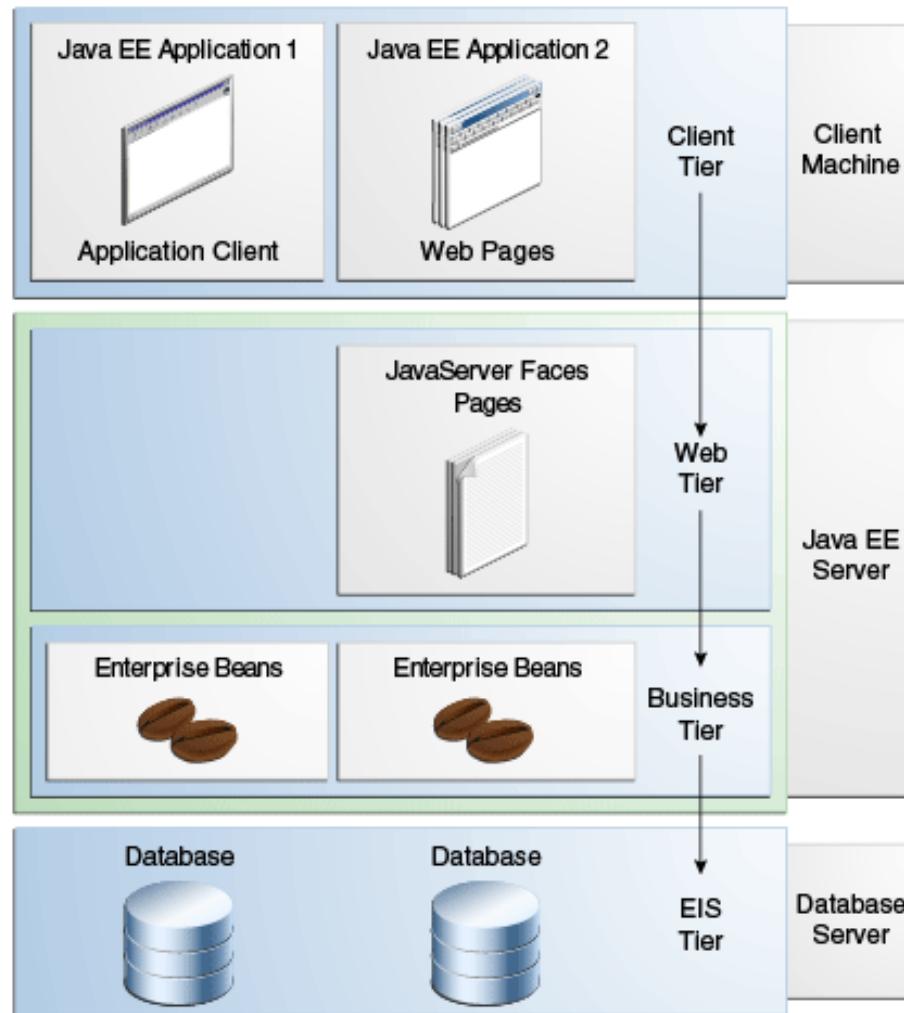


Arquitectura de Software



EJB Tier JavaEE

EJB en el Tier de Negocio de JavaEE



EB Enterprise Beans

- ▶ Son componentes portables y reusables de lado servidor que encapsulan lógica de negocio.
- ▶ Son una colección de archivos .java y XML empaquetados en una simple unidad EB modules.
- ▶ Se ejecutan dentro de un contenedor multihilo.
- ▶ El contenedor de EB proveen servicios de:
 - ▶ Seguridad
 - ▶ Transacciones
 - ▶ Persistencia
 - ▶ Servicio de Nombres



Cuando usar Enterprise Bean

- ▶ **La aplicación debe ser escalable**
 - ▶ Distribuir componentes de aplicación a través de varias máquinas
- ▶ **Las transacciones deben asegurar integridad de datos**
 - ▶ Acceso concurrente a objetos compartidos
- ▶ **La aplicación tiene una gran variedad de clientes**
 - ▶ reusabilidad

Enterprise Bean Tipos (I)

Enterprise Bean Type	Purpose
Session	Performs a task for a client; optionally, may implement a web service
Message-driven	Acts as a listener for a particular messaging type, such as the Java Message Service API

Enterprise Bean Session (I)

- ▶ Encapsulan servicios de lógica de negocios
- ▶ Coordina el trabajo de múltiples Entity JPA ó de otros Session
 - ▶ controladores
- ▶ Ejecutan operaciones de negocio de forma sincrónica y/ó asincrónica
- ▶ No son persistentes

Enterprise Bean Tipos (II)

Tipos Session beans :

- Stateful: mantiene en el servidor el estado de la sesión del cliente
- Stateless: no mantiene estado
- Singleton: existe un sólo bean para toda la aplicación o sea para todos los clientes por lo que soporta acceso concurrente

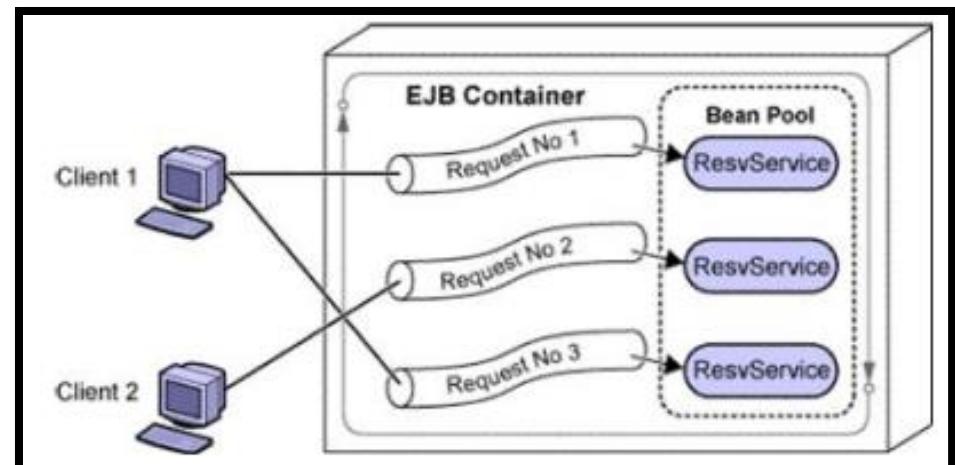
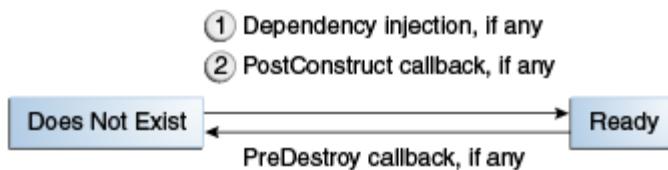
Tipos de Session Bean

- ▶ Tipos:
 - ▶ Stateful: mantienen estado conversacional
 - ▶ Stateless: no mantienen estado conversacional
- ▶ El estado de un objeto consiste de los valores de sus variables de instancia.
 - ▶ En el ejemplo servicioReserva tiene una reservación
- ▶ Estado conversacional se refiere al hecho que el cliente interactúa con su bean



Stateless Session Bean

- ▶ No se mantiene estado con el cliente
 - ▶ Cada request se deben pasar parámetros
- ▶ Los valores del bean existen sólo durante la invocación
- ▶ Pueden soportar múltiples clientes por lo que ofrecen mejor escalabilidad



Stateful Session Bean

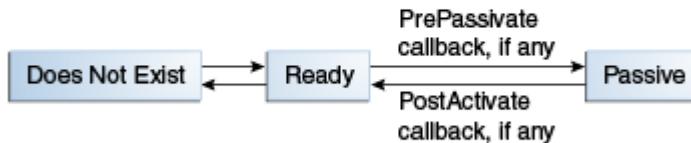
- ▶ El estado se retiene para la duración de la sesión
- ▶ Si el cliente remueve el bean ó termina su ciclo de vida la sesión finaliza y el estado desaparece

① Create

② Dependency injection, if any

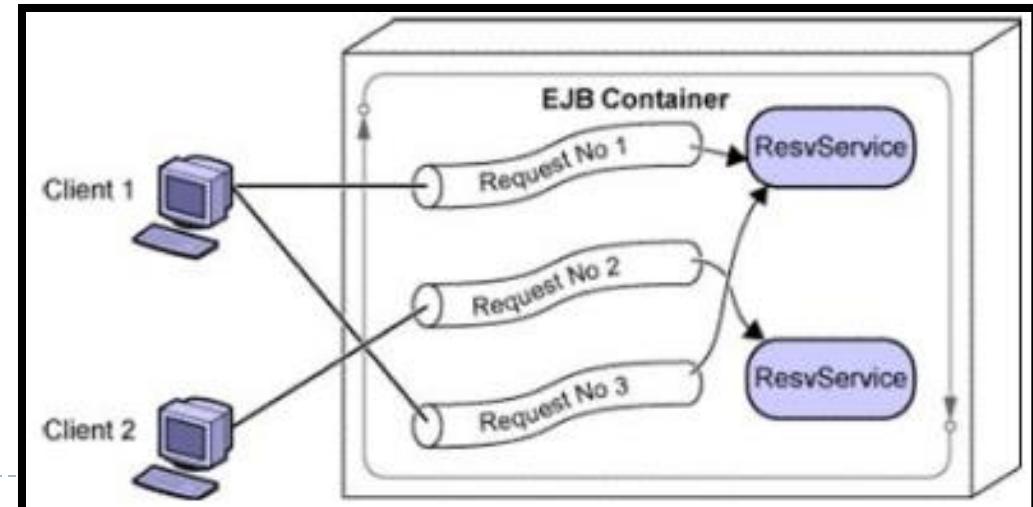
③ PostConstruct callback, if any

④ Init method, or ejbCreate<METHOD>, if any

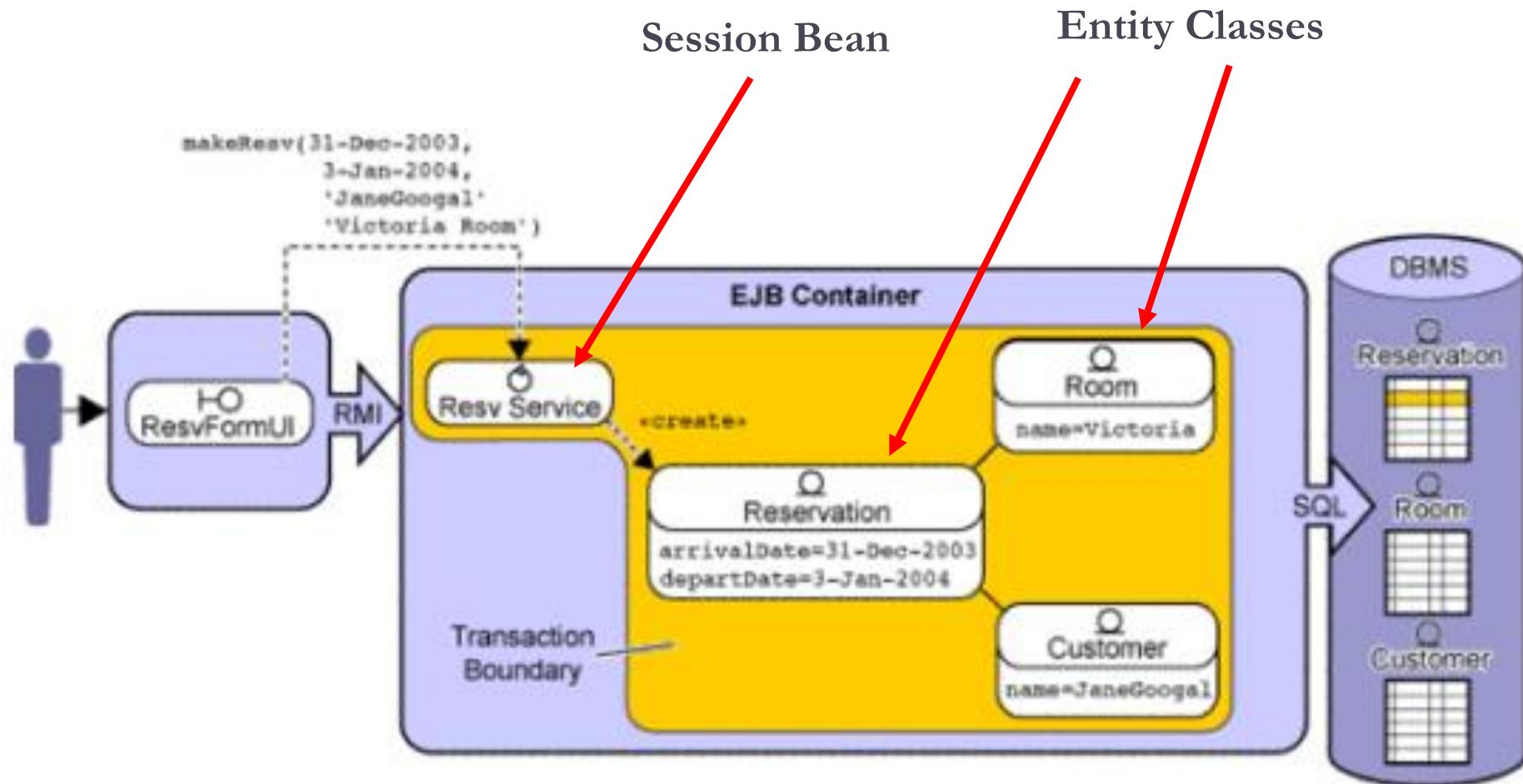


① Remove

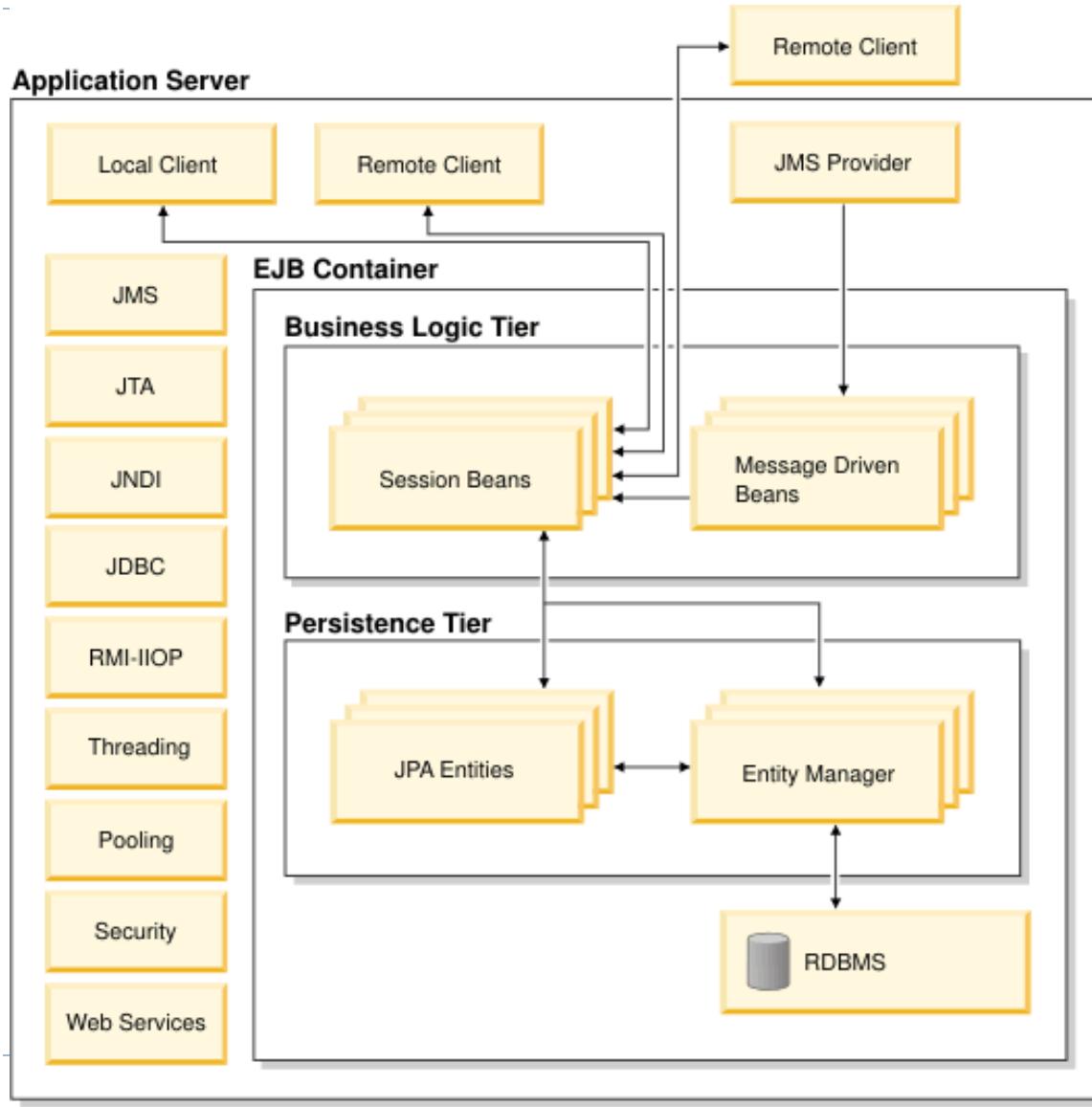
② PreDestroy callback, if any



Enterprise Bean y Elementos MVC



Arquitectura EJB 3.0



Transacciones en Session Bean

- ▶ Los métodos de los Session EJB se ejecutan como una simple transacción que al final realiza commit ó rollback.
- ▶ El usuario no requiere insertar código de manejo de transacciones

Clientes de EJB

- Web
- Aplicación Cliente (swing, console)
- Otro EB
- Servicios Web

Clientes de EJB

- Clientes en la misma unidad de despliegue
 - Sin Interfaces
- Clientes en diferentes unidades de despliegue
 - Con Interfaces

Bean

- ▶ Objeto gestionado por el contenedor Java EE
- ▶ Servicios adicionales dados por el contenedor a un bean
- ▶ En Java EE la mayoría de los objetos son bean
- ▶ A excepción de las entidades de JPA



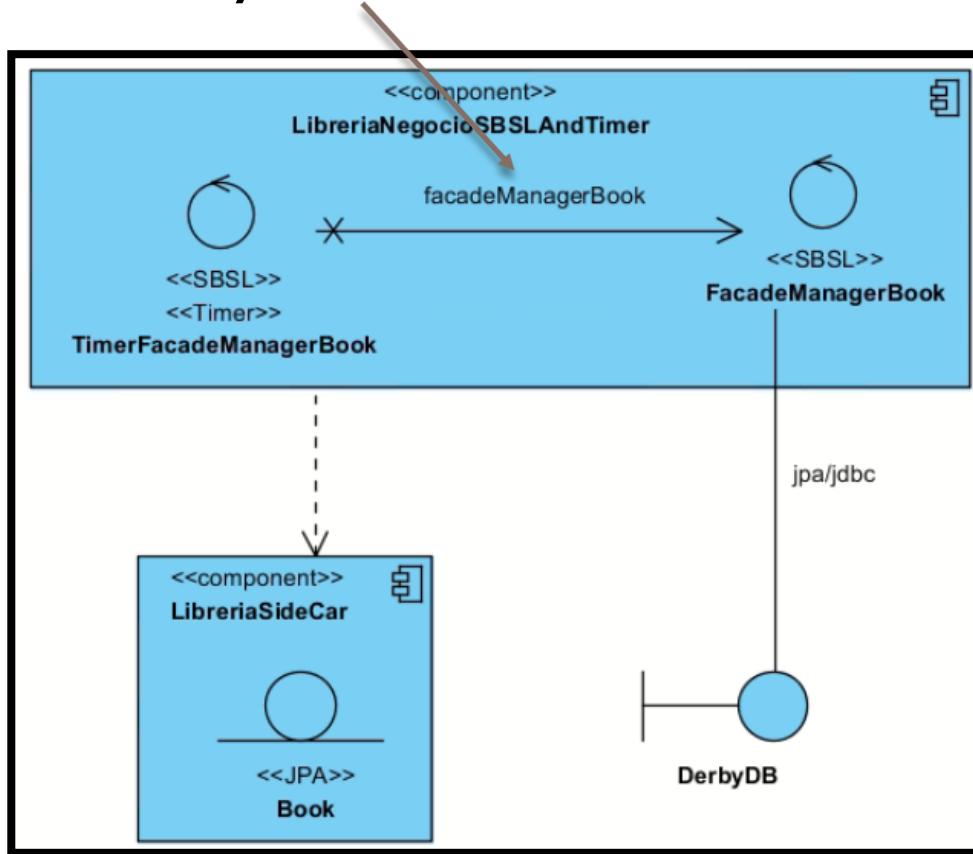
Inyección de Dependencias

- ▶ El objetivo principal de Inyección de Dependencias es eliminar dependencias de una aplicación.
 - ▶ Esto hace que el sistema sea más desacoplado y mantenible
- ▶ Los beans dependen de otros beans
- ▶ Los beans no crean sus propias dependencias
- ▶ Servicio prestado por el contenedor
 - ▶ Soporta inversión de control (IoC)
 - ▶ El control lo tiene la plataforma



CDI

- ▶ Context and Dependency Injection
- ▶ Contenedor inyecta referencias de bean en otros beans



Inyección de Dependencias

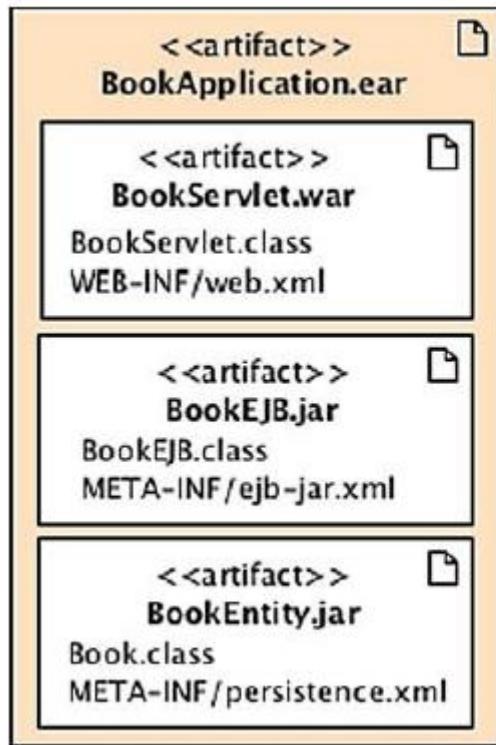
- ▶ La CDI permite a los componentes acceder varias clases de recursos a través del contenedor mediante un modelo de “push” que es definido en tiempo de despliegue.
- ▶ No se requiere un acceso directo (usando new)
- ▶ Ejemplo: un cliente de un EJB usa DI para referenciar la interfaz local de dicho EJB usando @EJB o @Inject

```
@EJB  
private static BookEJB bookEJB;
```

```
@Inject  
private static BookEJB bookEJB;
```

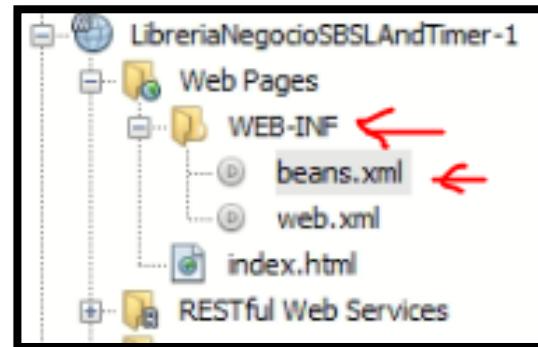
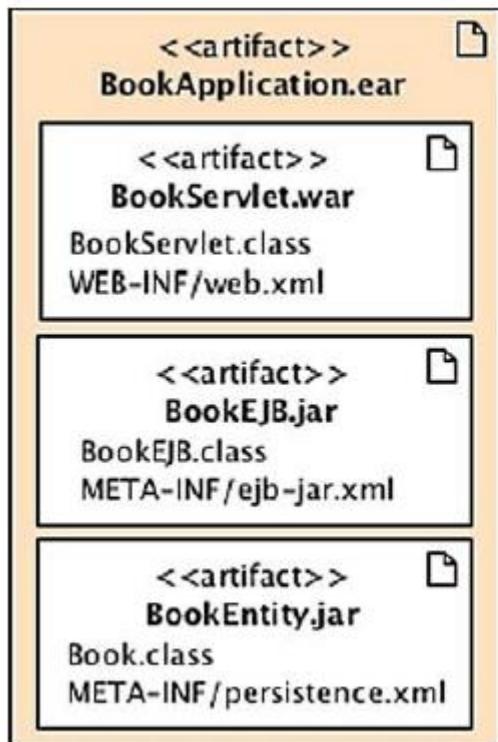
Empaquetar los EJB

- Los EJB se pueden usar en los containeres Web o EJB
- se pueden empaquetar en JAR, WAR, EAR



Empaquetar los EJB

Para habilitar el CDI se agrega un archivo beans.xml que le permite al contenedor buscar clases instanciables dentro de la unidad de despliegue



```
<beans xmlns:xsi="..."  
       bean-discovery-mode="all">  
</beans>
```

beans.xml

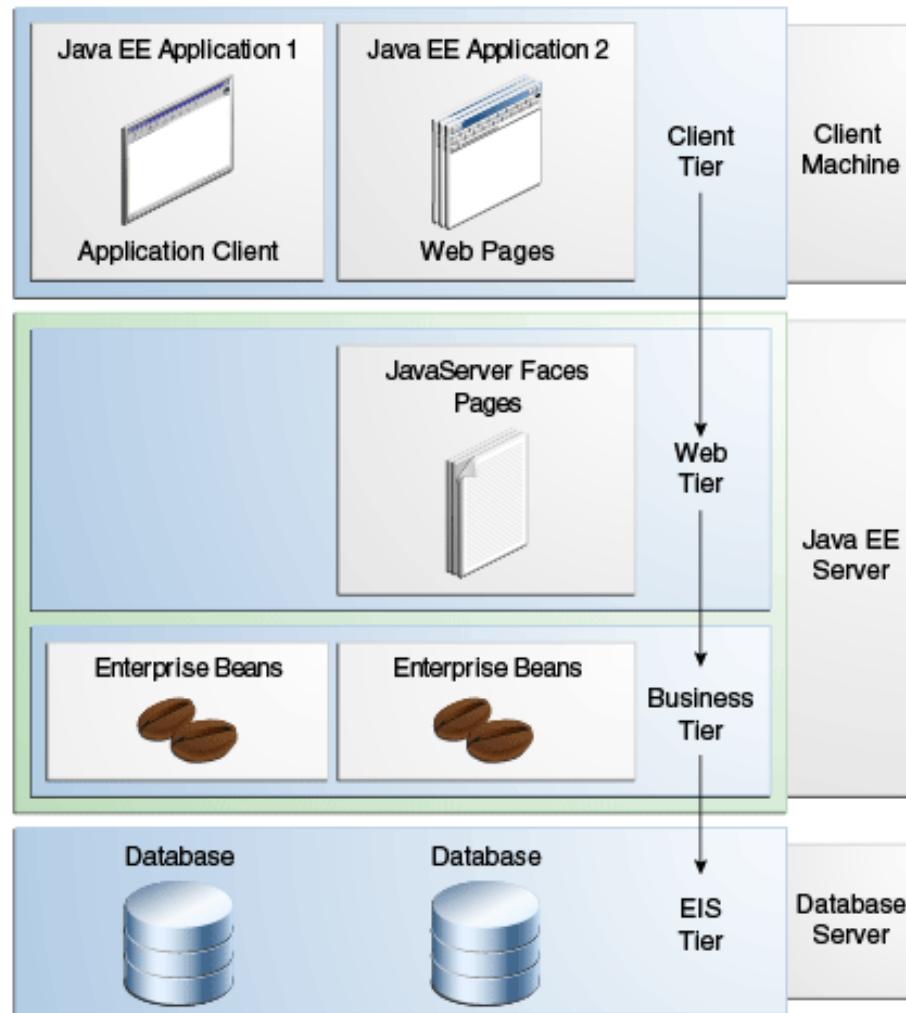


Arquitectura de Software



EJB Tier JavaEE

EJB en el Tier de Negocio de JavaEE



EB Enterprise Beans

- ▶ Son componentes portables y reusables de lado servidor que encapsulan lógica de negocio.
- ▶ Son una colección de archivos .java y XML empaquetados en una simple unidad EB modules.
- ▶ Se ejecutan dentro de un contenedor multihilo.
- ▶ El contenedor de EB proveen servicios de:
 - ▶ Seguridad
 - ▶ Transacciones
 - ▶ Persistencia
 - ▶ Servicio de Nombres

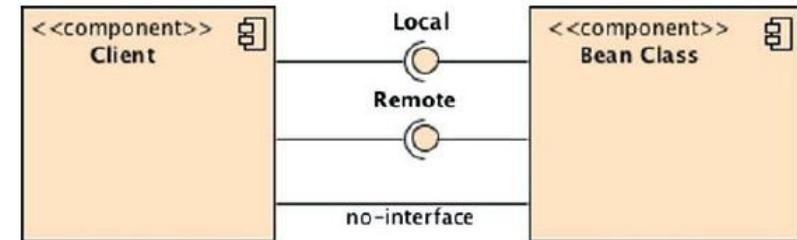
EB Enterprise Beans

- ▶ Clientes en otra unidad de despliegue

Elementos de Session Bean

▶ Remote Interface

- ▶ define los servicios públicos que expone el componente a los clientes remotos
 - ▶ Clientes en máquinas diferentes
 - ▶ Clientes en JVM diferentes
 - ▶ Paso por valor



▶ Local Interface

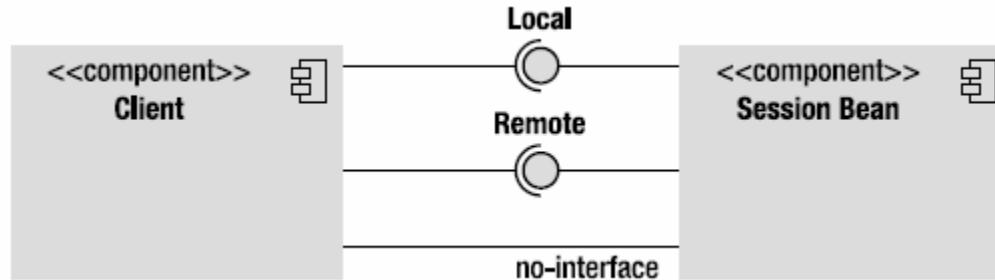
- ▶ define los servicios públicos que expone el componente a los clientes locales
 - ▶ Clientes en la misma JVM
 - ▶ Paso por referencia

▶ Enterprise Bean Class

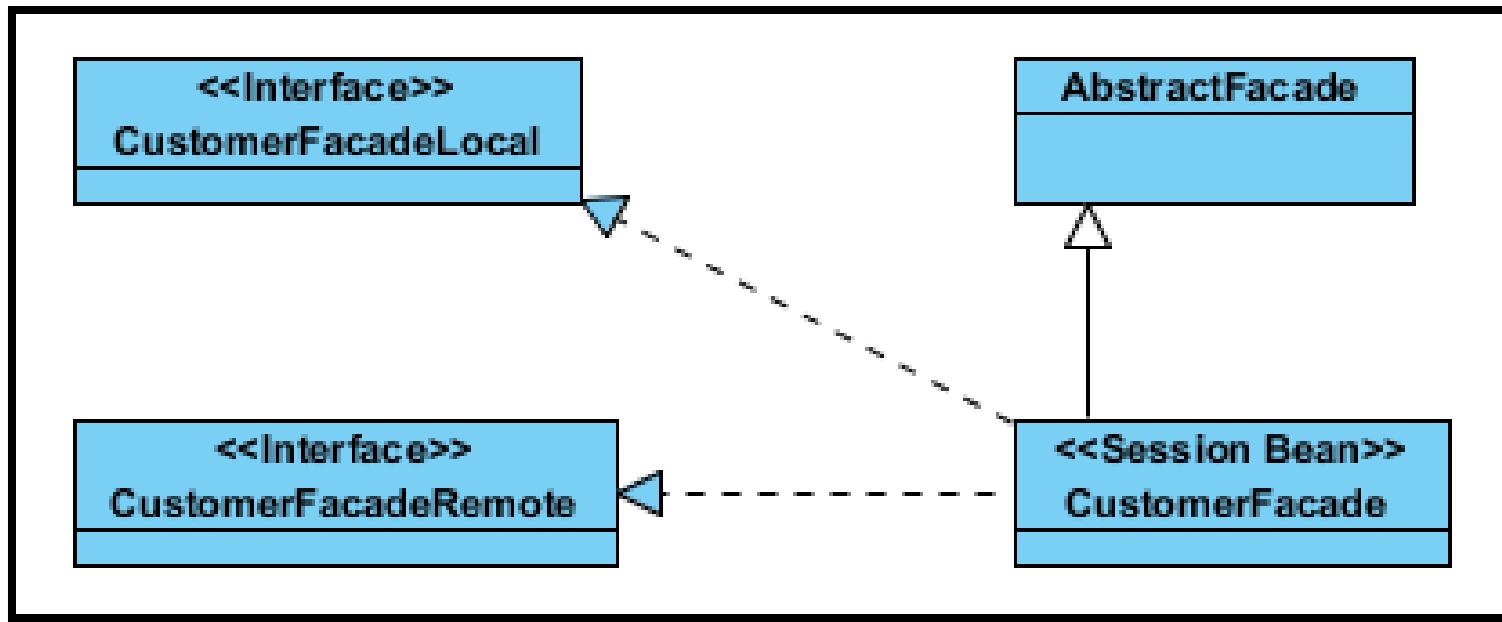
- ▶ provee la implementación (código java) de la interfaz

Interfaces de EJB

- La vista de “no-interface” es una variación de la vista local que expone todos los métodos públicos de negocio de manera local sin exponer interfaces.
- Los clientes que realizan llamadas remotas usan RMI

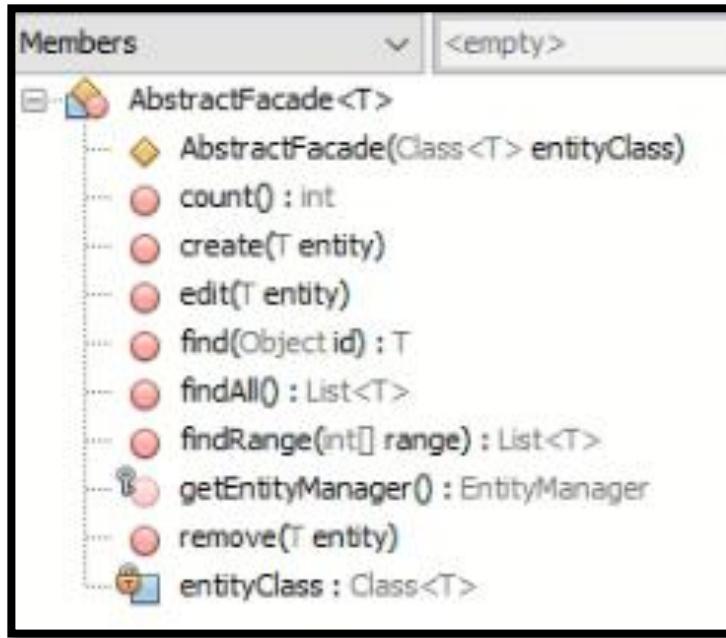


Elementos de Session Bean

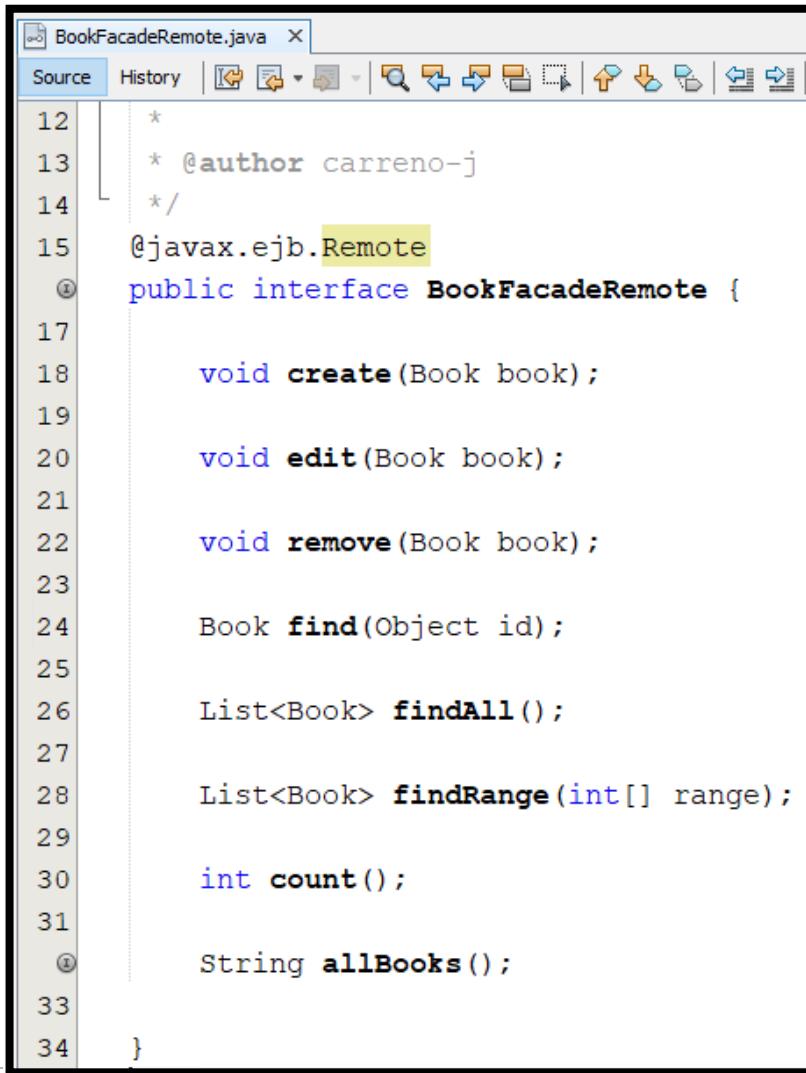


Un bean puede exponer la interface remota y la local mediante una abstractfacade

Elementos de Session Bean

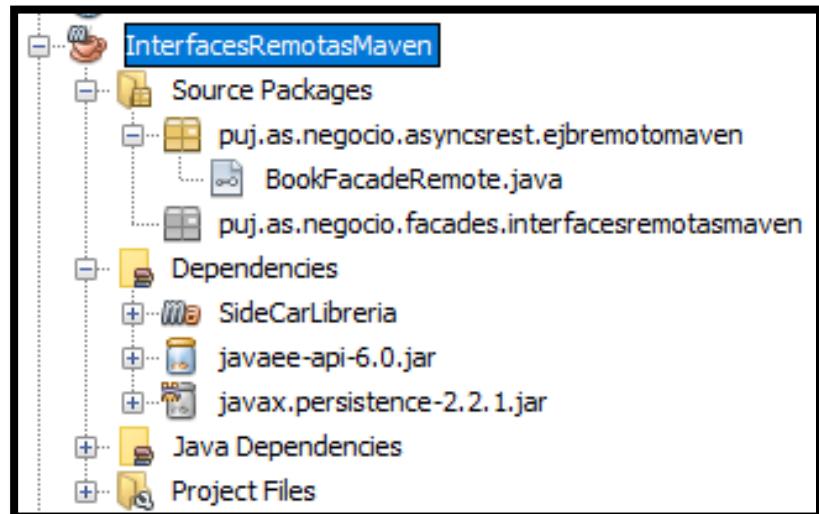


Elementos de Session Bean -Interface Provista-



```
12  *
13  * @author carreno-j
14  */
15  @javax.ejb.Remote
16  public interface BookFacadeRemote {
17
18      void create(Book book);
19
20      void edit(Book book);
21
22      void remove(Book book);
23
24      Book find(Object id);
25
26      List<Book> findAll();
27
28      List<Book> findRange(int[] range);
29
30      int count();
31
32      String allBooks();
33  }
```

•Las interfaces quedan en su propia unidad de despliegue



Clientes de EJB

- Web
- Aplicación Cliente (swing)
- Otro EB
- Servicios Web

Accediendo a los EJB desde los clientes

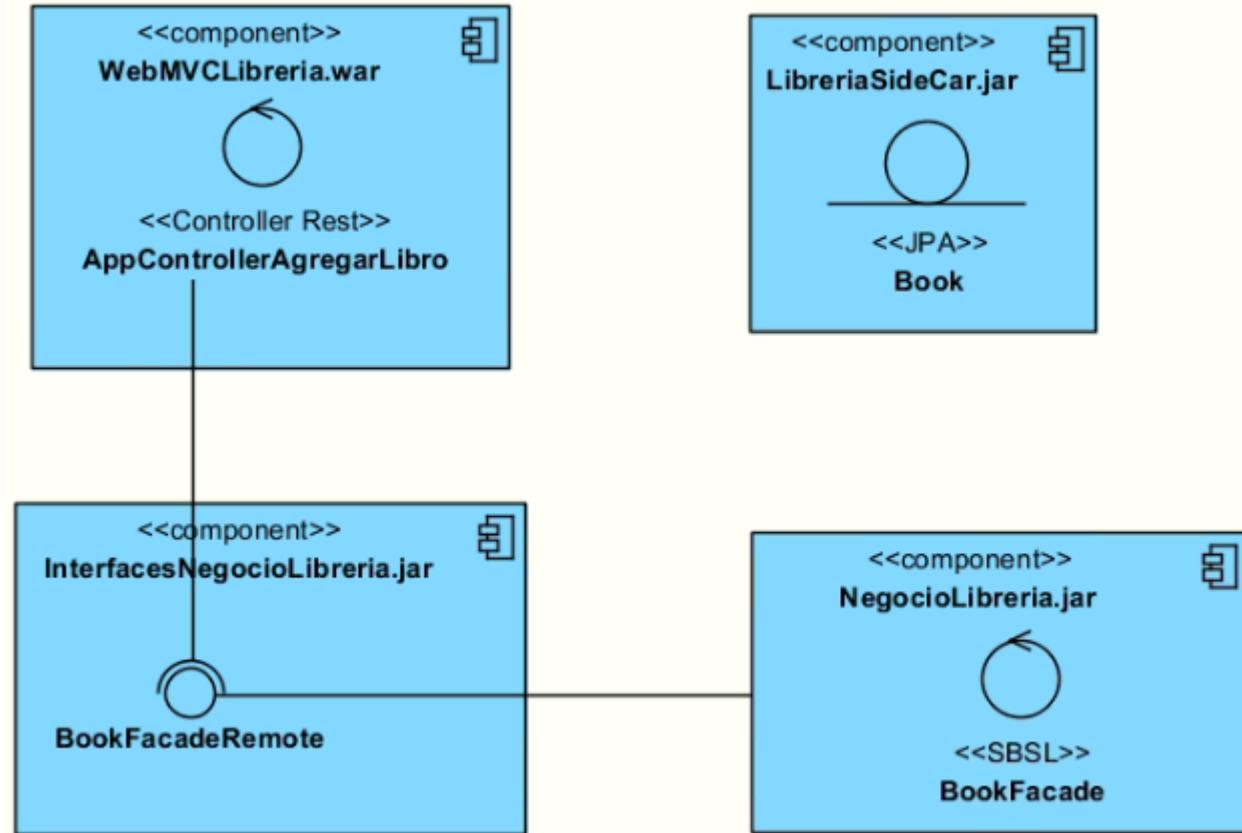
- ▶ El cliente de un EJB obtiene una referencia a una instancia de un EJB usando:
 - ▶ Dependency Injection: para clientes que se ejecutan dentro de:Java EE server-managed environment, JavaServer Faces web applications, JAX-RS web services, other enterprise beans, or Java EE application clients support dependency injection using the javax.ejb.EJB annotation.
 - ▶ JNDI Lookup: aplicaciones que se ejecutan por fuera de un Java EE server-managed environment



Interfaces Provistas EJB

@EJB

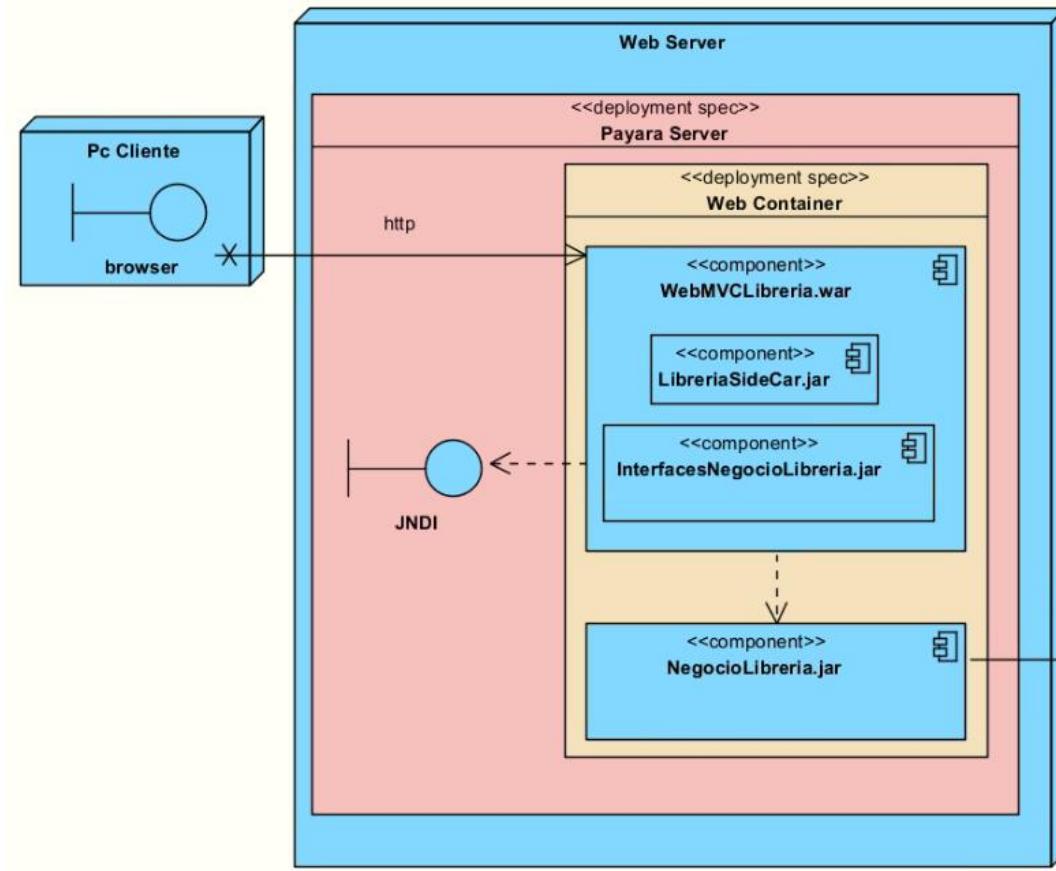
```
private BookFacadeRemote booksFacade;
```



Acceso vía Interface Remota Mismo Servidor Diferente Unidad de despliegue

@EJB (ó @Inject Usando JNDI)

```
private BookFacadeRemote booksFacade;
```

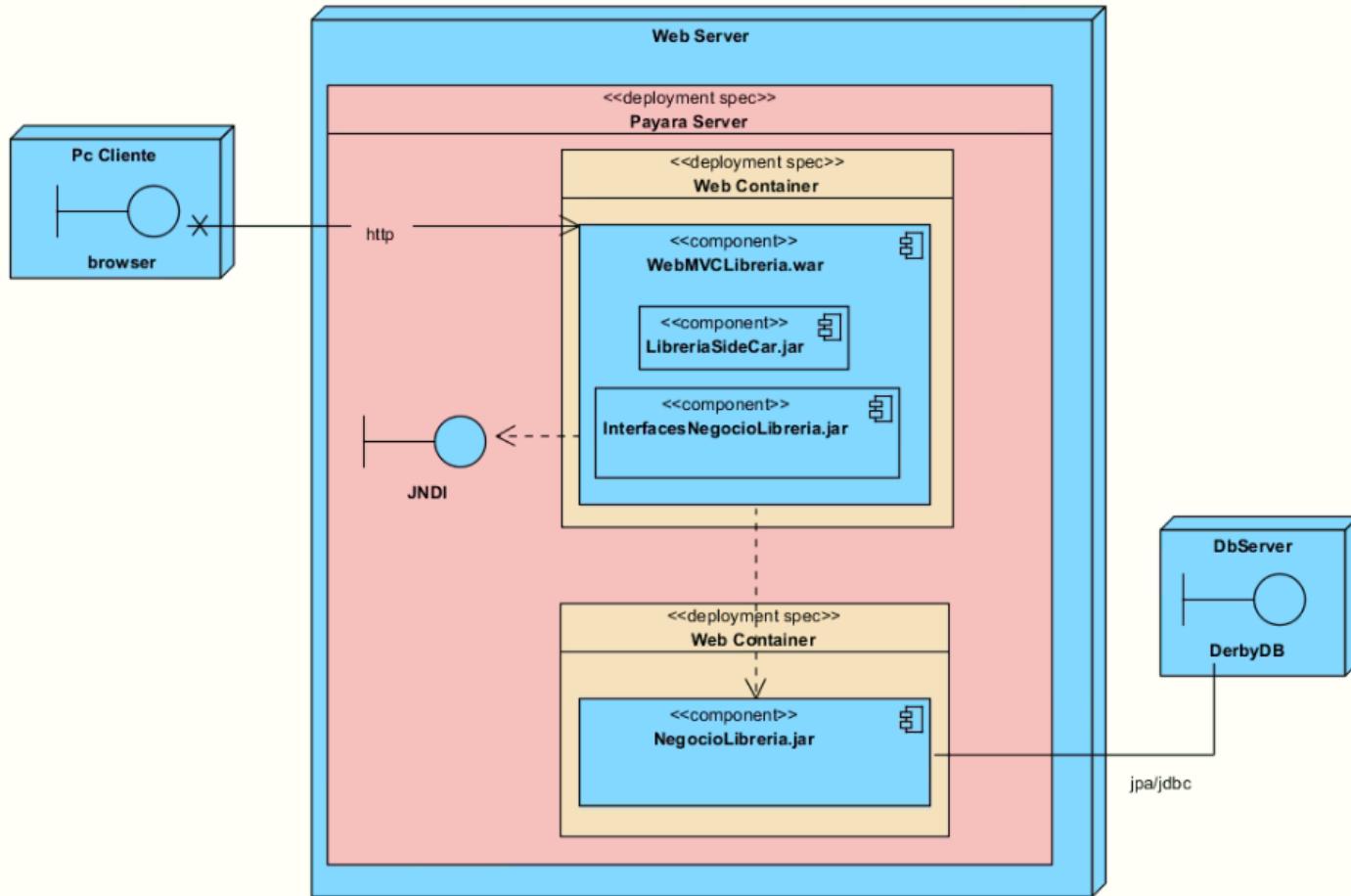


- El cliente debe usar la unidad de despliegue donde están las interfaces

Acceso vía Interface Remota Diferentes Container Mismo Servidor

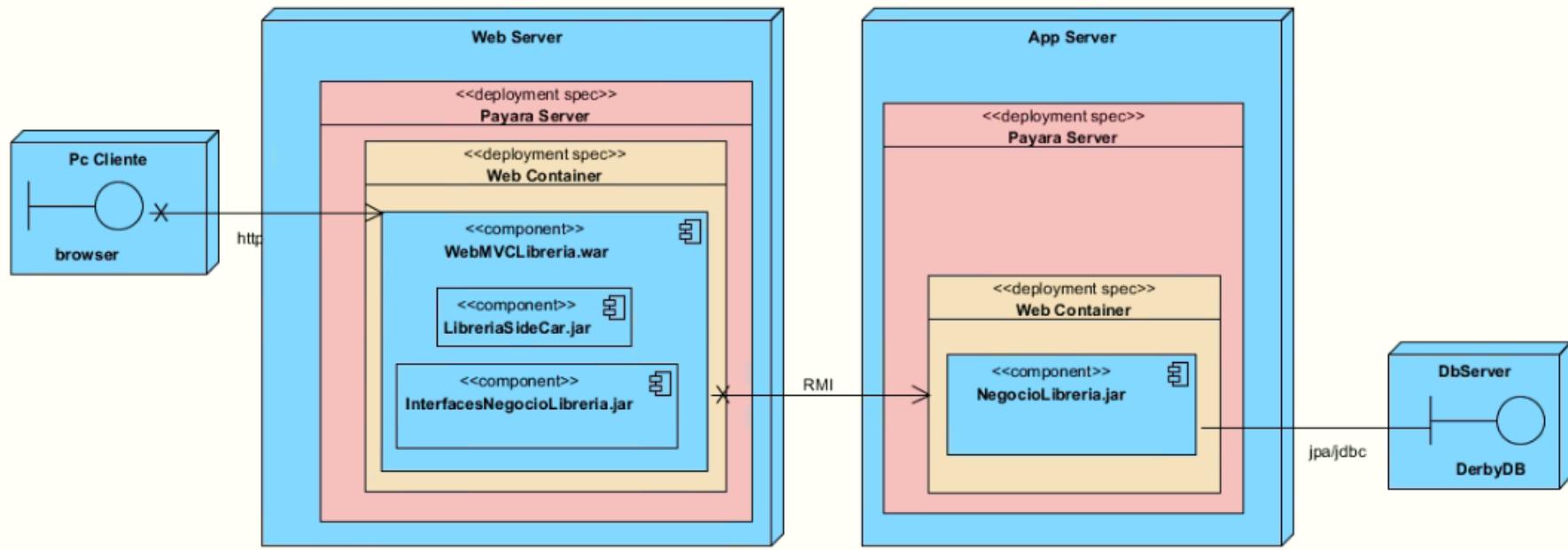
@EJB

private BookFacadeRemote booksFacade;



Acceso vía Interface Remota Diferentes Servidores

NO SE USA @EJB



Acceso Remoto Diferentes Servidores

- ▶ Código del cliente que accede un EJB Remoto en otra máquina usando JNDI Lookup

```
public void DistributedEJBConnection() {  
    System.out.println("test Remote FULL-->");  
    try {  
        String host="localhost";// if you run your client and server sample on same machine  
        String port ="3700";//default  
        // to obtain port use asadmin get "configs.config.server-config.iiop-service.iiop-listener.orb-listener-1.*"  
        Properties prop = new Properties();  
        prop.put("org.omg.CORBA.ORBInitialHost",host);  
        prop.put("org.omg.CORBA.ORBInitialPort",port);  
        InitialContext context =new InitialContext(prop);  
        BookFacadeRemote facade =(BookFacadeRemote)  
            context.doLookup("java:global/ejbRemotoMaven-1/BooksFacade  
            //!puj.as.negocio.asyncsrest.ejbremotomaven.BookFacadeRemote")  
            context.doLookup("java:global/WSLibreria-1/BookFacade"  
                + "!puj.as.negocio.servicios.wslibreria.facades.BookFacade")  
            ;  
        String text= facade.allBooks();  
        System.out.println(text);  
        context.close();  
    } catch(Exception e) {  
        e.printStackTrace();  
    }  
}
```

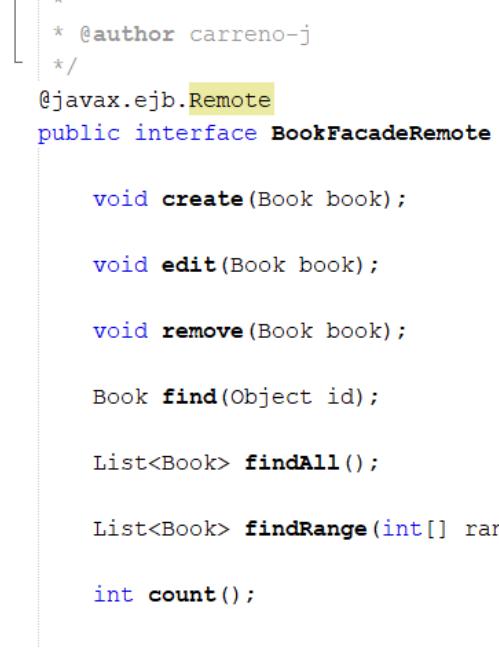
Acceso Remoto Diferentes Servidores

- ▶ Siempre que se despliega un EJB se le asigna un nombre dentro del JNDI:
- ▶ Portable JNDI names for EJB BookFacade:
[java:global/WSLibreria-I/BookFacade,
java:global/WSLibreria-
I/BookFacade!puj.as.negocio.servicios.wslibreria.faca
des.BookFacade]



EJB vs JAX RS (Rest)

- ▶ ¿En donde el JAX RS REST deja la definición de la interfaz provista? ¿Nombre y extensión del archivo?



The screenshot shows a Java code editor with the file `BookFacadeRemote.java` open. The code defines a remote interface for managing books. It includes methods for creating, editing, removing, finding, and counting books, as well as a method to get all books. Annotations like `@author` and `@javax.ejb.Remote` are present. The code is color-coded for readability.

```
12  *
13  * @author carreno-j
14  */
15  @javax.ejb.Remote
16  public interface BookFacadeRemote {
17
18      void create(Book book);
19
20      void edit(Book book);
21
22      void remove(Book book);
23
24      Book find(Object id);
25
26      List<Book> findAll();
27
28      List<Book> findRange(int[] range);
29
30      int count();
31
32      String allBooks();
33
34  }
```

CRUD	REST	
CREATE	POST	Create a (sub)resource
RETRIEVE	GET	Retrieve a representation of a resource
UPDATE	PUT	Modify a resource/create a new resource
DELETE	DELETE	delete a resource
	OPTIONS	Discover what HTTP methods are supported by the resource
	HEAD	requests headers only (similar to GET but omits representation)

Bibliografia

<http://docs.oracle.com/javaee/7/tutorial/doc/home.htm>



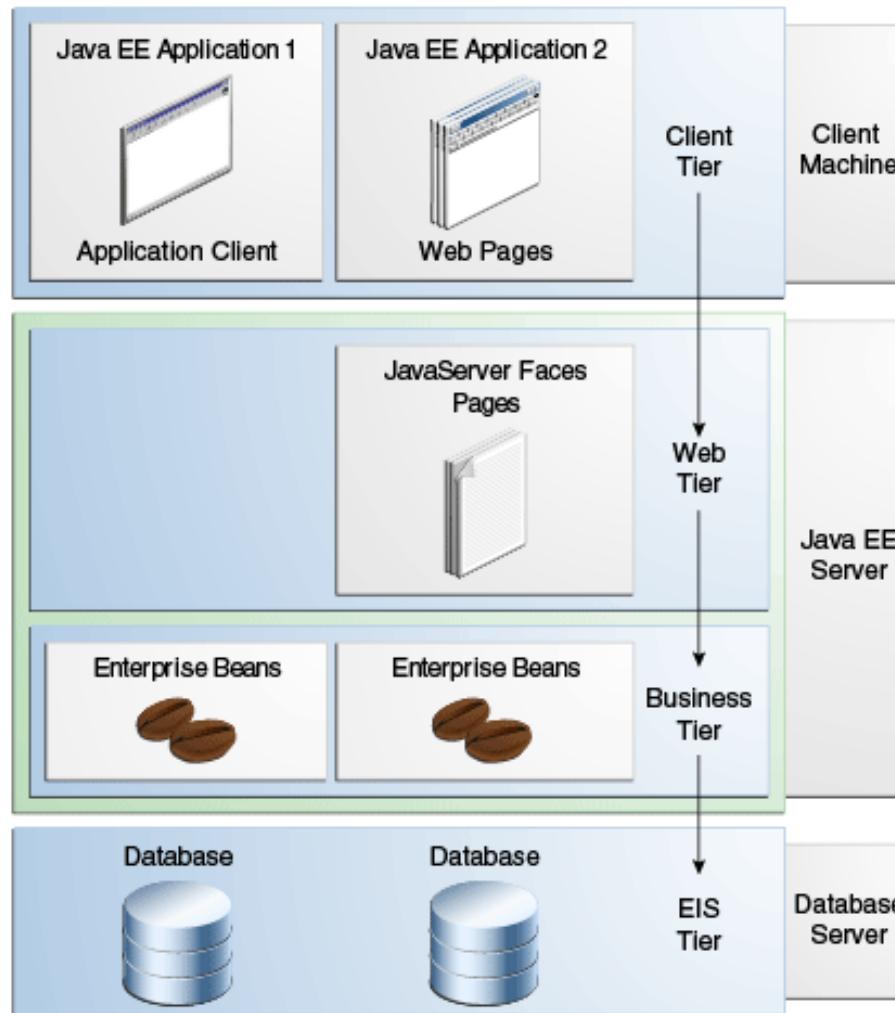
Arquitectura de Software

Julio Carreño



EJB MDB

Aplicaciones MultiTier Distribuidas(1)

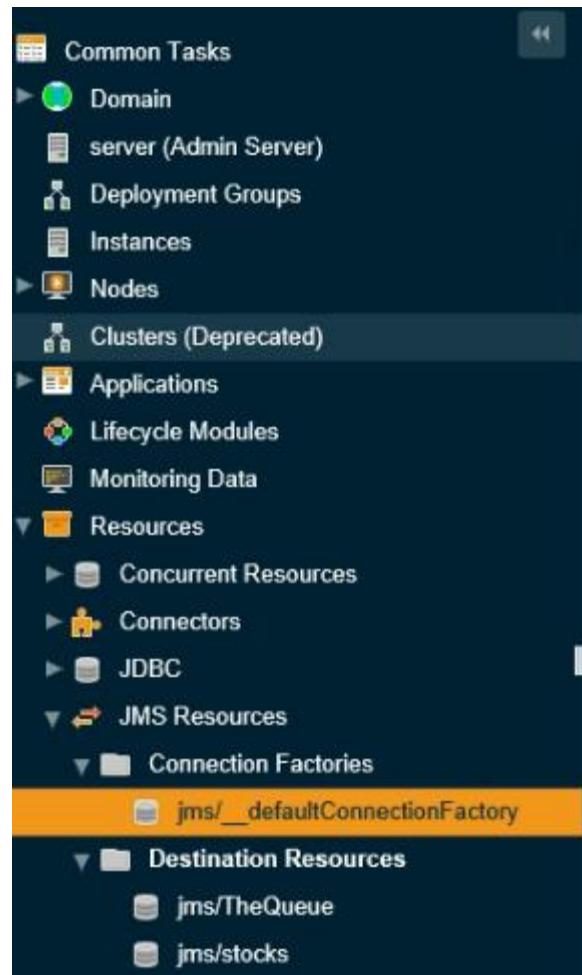


JMS (Java Message Service)

- ▶ El JMS permite acceder sistemas de mensajería (MOM)
- ▶ El JMS API permite a las aplicaciones crear, enviar, recibir y leer mensajes
- ▶ El JMS habilita la comunicación:
 - ▶ Con pérdida de acoplamiento
 - ▶ Asincrónica
 - ▶ Confiable
 - ▶ El mensaje se envía exactamente una sola vez

Open MQ y JMS

- ▶ Open MQ implementa el estándar (JMS), no tiene que configurar un intermediario JMS externo (por ejemplo, ActiveMQ, RabbitMQ ..) para este ejemplo y puede utilizar la versión embebida (piénselo dos veces si lo utiliza en producción).

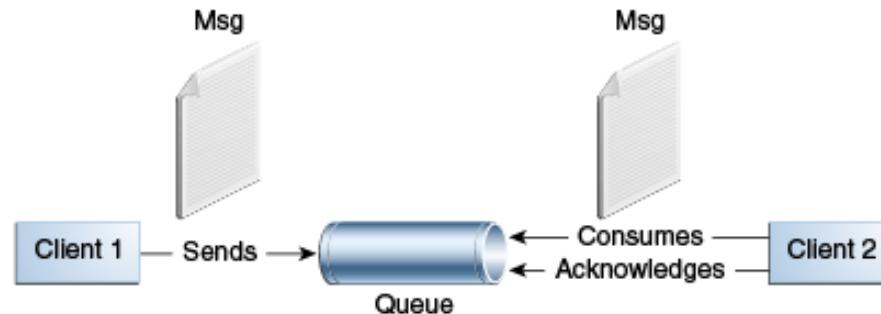


Aproximaciones de mensajería

- ▶ Punto a punto
- ▶ Publish subscribe

Punto a Punto

- ▶ Usa colas
- ▶ Cada mensaje es enviado a una cola específica
- ▶ Los clientes extraen los mensajes desde la cola establecida para guardar sus mensajes
- ▶ Cada mensaje tiene sólo **un consumidor**
- ▶ No hay restricciones de tiempo para enviar ó recibir
- ▶ El receptor informa a la cola del procesamiento exitoso del mensaje



Punto a Punto



Figure 13-3. P2P model

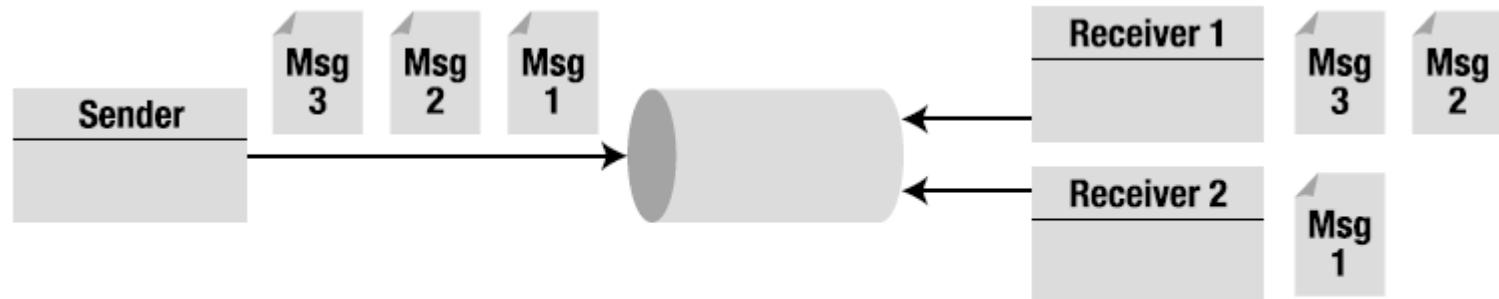
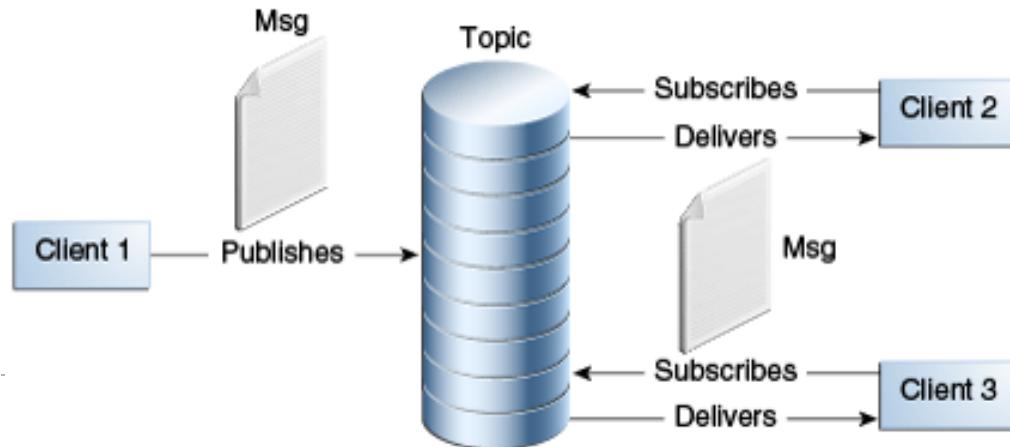


Figure 13-4. Multiple receivers

Publish Subscribe

- ▶ Usa topics
- ▶ Publicadores y suscriptores son anónimos y pueden de manera dinámica publicar ó subscribirse a contenido
- ▶ Cada mensaje puede tener **múltiples consumidores**
- ▶ Un cliente que se subscribe a un tópico puede consumir sólo mensajes publicados después que el cliente ha creado una suscripción
- ▶ JMS puede recibir mensajes mientras no existan suscriptores activos



Publish Subscribe

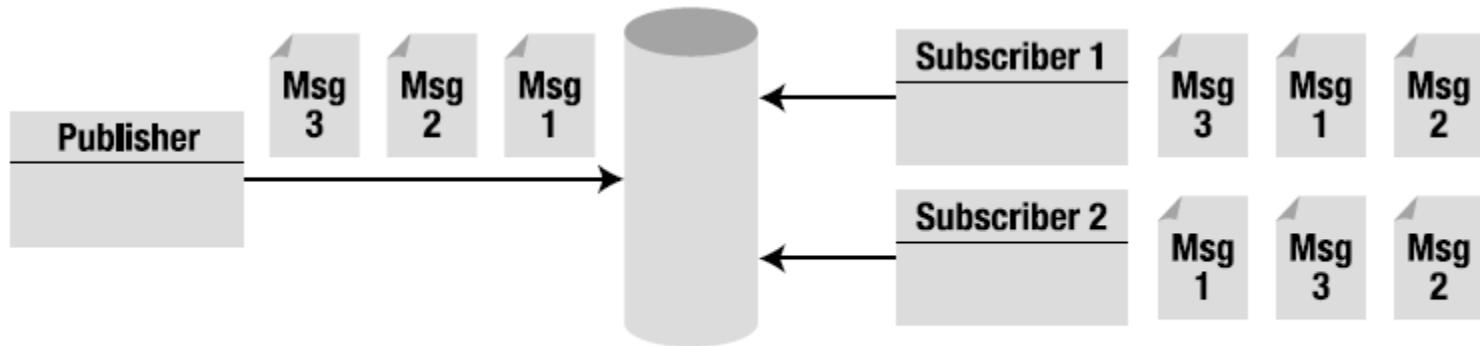
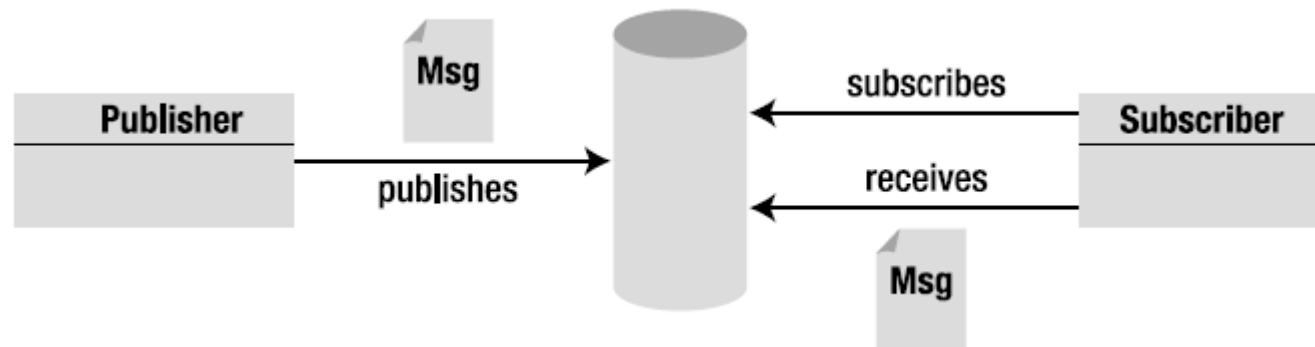


Figure 13-6. Multiple subscribers

JMS consumiendo mensajes

- ▶ Existen dos tipos de consumo:
 - ▶ Consumo sincrónico
 - ▶ Consumo asincrónico

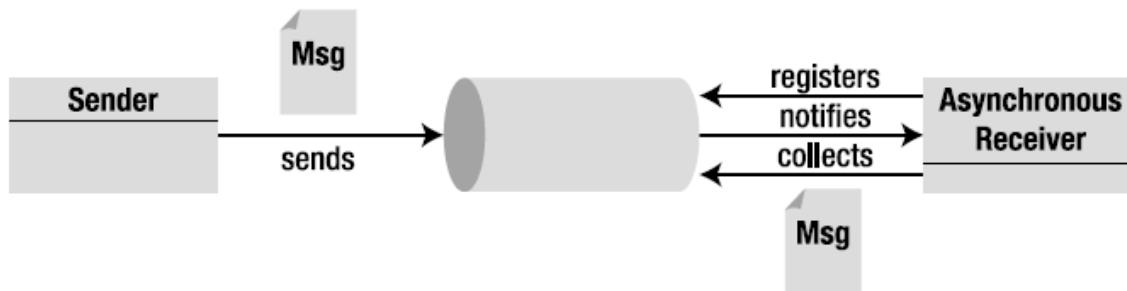
JMS consumiendo mensajes

- ▶ Consumo sincrónico: el consumidor explícitamente recoge el mensaje desde el destino (cola/topic) usando un ciclo infinito que espera por llegada de mensajes



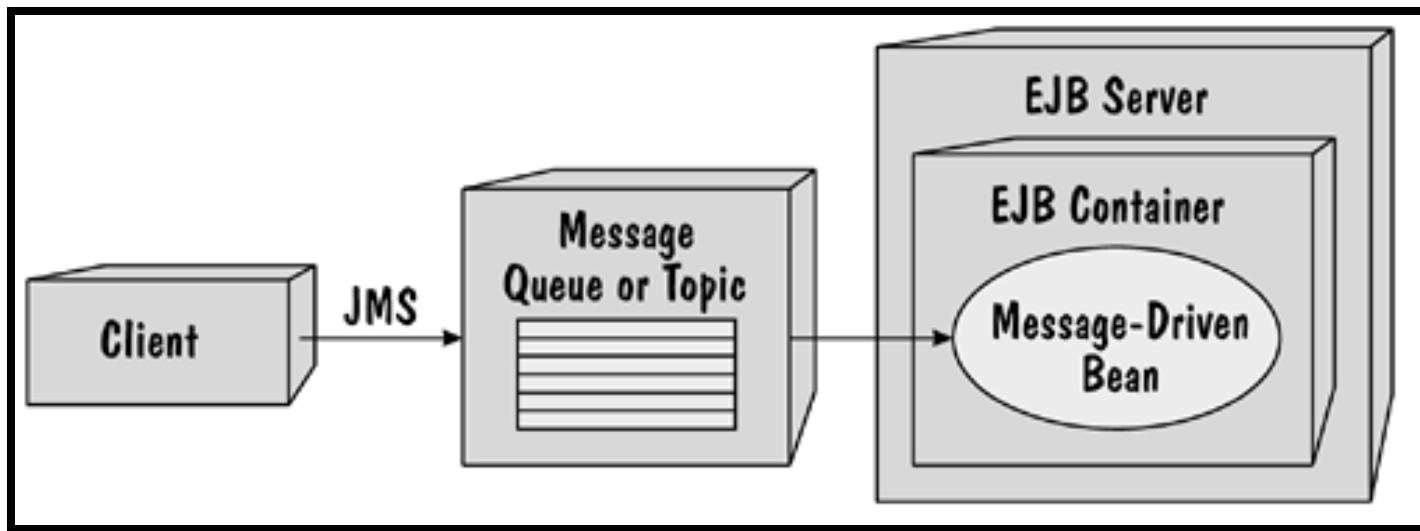
JMS consumiendo mensajes

- ▶ Consumo asincrónico: el consumidor se registra a un evento que es disparado cuando llega un mensaje. Debe implementar la interface MessageListener y cuando el mensaje llega se invoca un método onMessage() en el consumidor



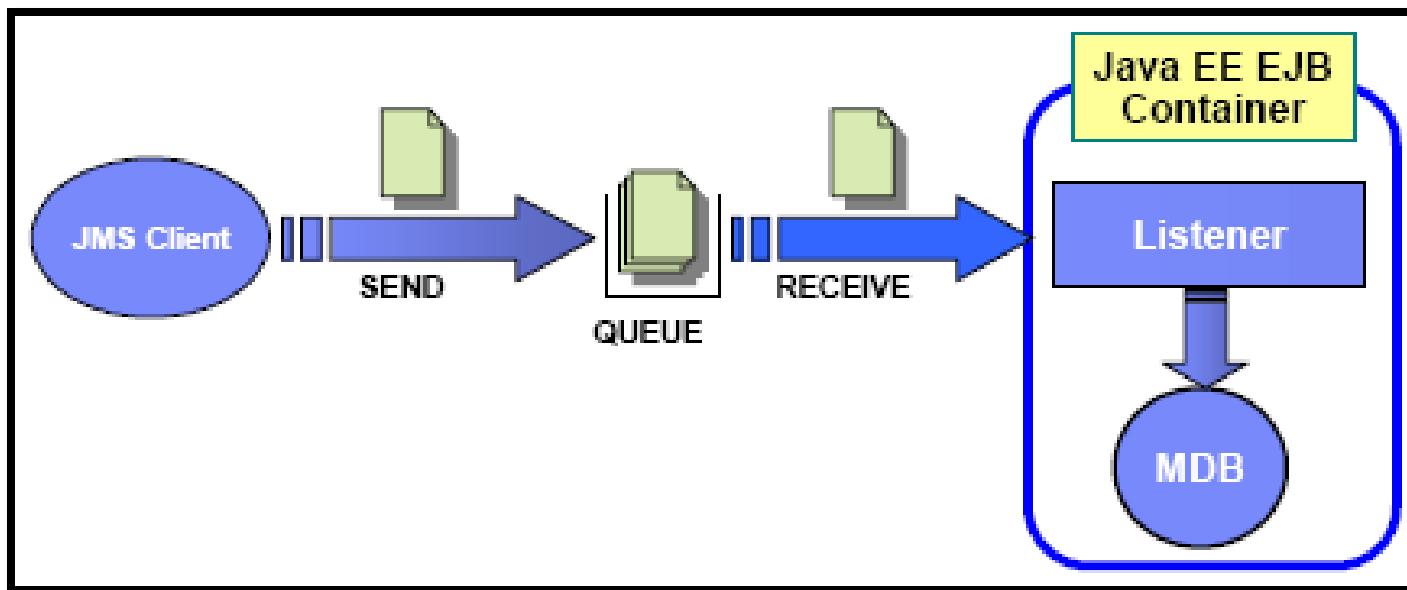
MDB Message Driven Bean

- ▶ Permite a las aplicaciones procesar (consumir) mensajes **asincrónicamente**



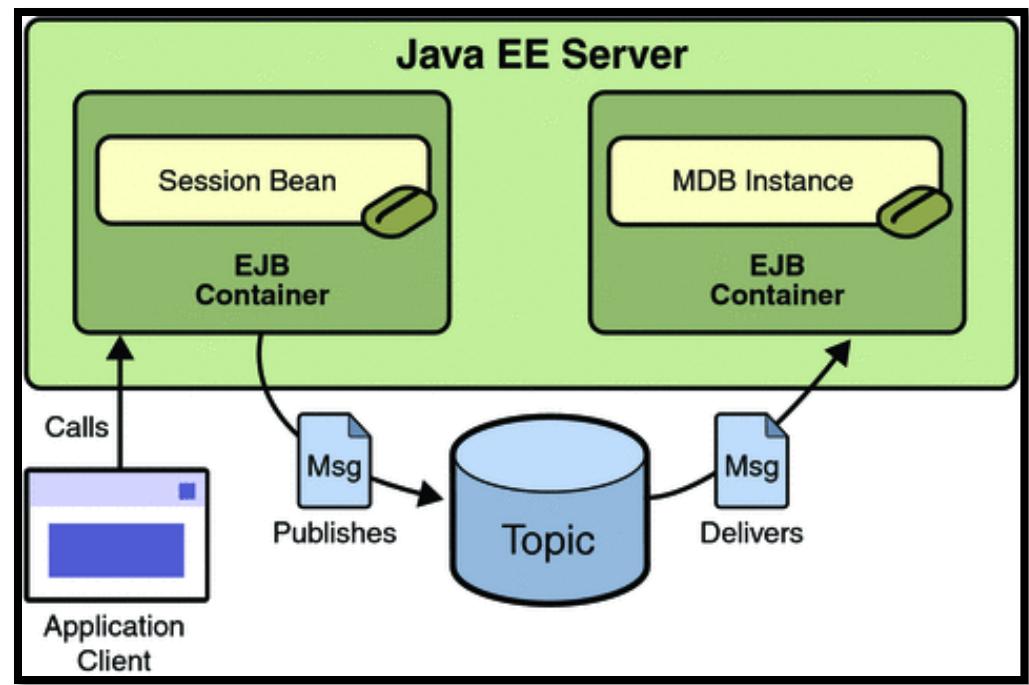
MDB Message Driven Bean

- ▶ Actúan como un “listener” para un determinado tipo de mensajería como el JMS (Java Message Service API)



MDB Message Driven Bean

- ▶ Los mensajes pueden ser enviados por:
 - ▶ cualquier componente JavaEE
 - ▶ Una aplicación cliente
 - ▶ Un EJB
 - ▶ Un web component
 - ▶ Ó cualquier aplicación ó sistema que no use JavaEE



MDB vs Session Beans

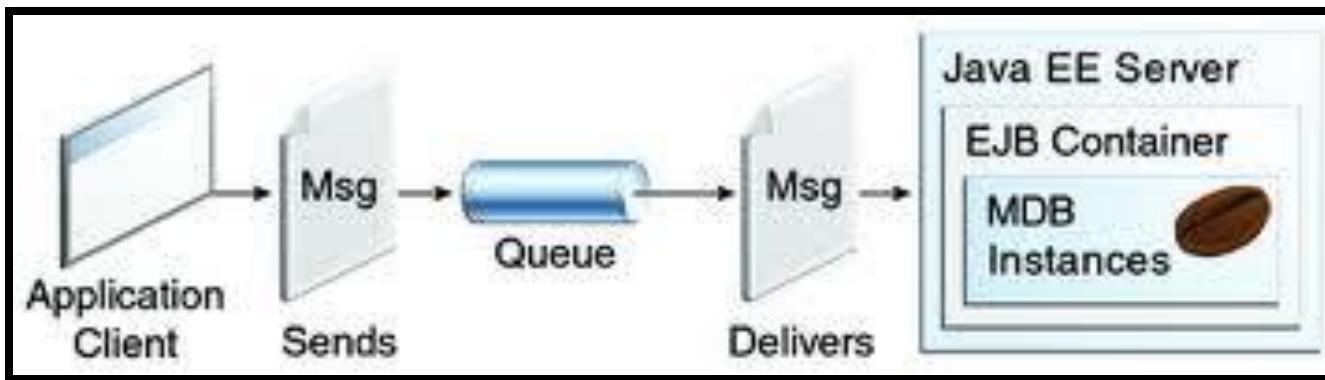
- ▶ En los MDB el cliente no accede a través de interfaces
- ▶ El MDB sólo se compone del bean class
- ▶ Se parece en cierto modo al stateless session bean en:
 - ▶ No retiene estado conversacional
 - ▶ Todas las instancias son equivalentes, de modo que el procesamiento de un mensaje se puede asignar a cualquier instancia MDB
 - ▶ Pool de MDB que permite el procesamiento concurrente
 - ▶ Un solo MDB puede procesar mensajes desde múltiples clientes

MDB vs Session Beans

- ▶ Los beans de sesión le permiten enviar y recibir mensajes de forma sincrónica, pero no de forma asincrónica.
- ▶ Los MDB pueden consumir de forma confiable mensajes desde una cola ó una suscripción

Clientes MDB

- ▶ Los clientes no localizan ni tampoco invocan métodos directamente de los MDB
- ▶ El cliente accede el MDB a través de un broker de mensajes (JMS) enviando mensajes a un destinatario para el cual la clase MDB tiene un escucha

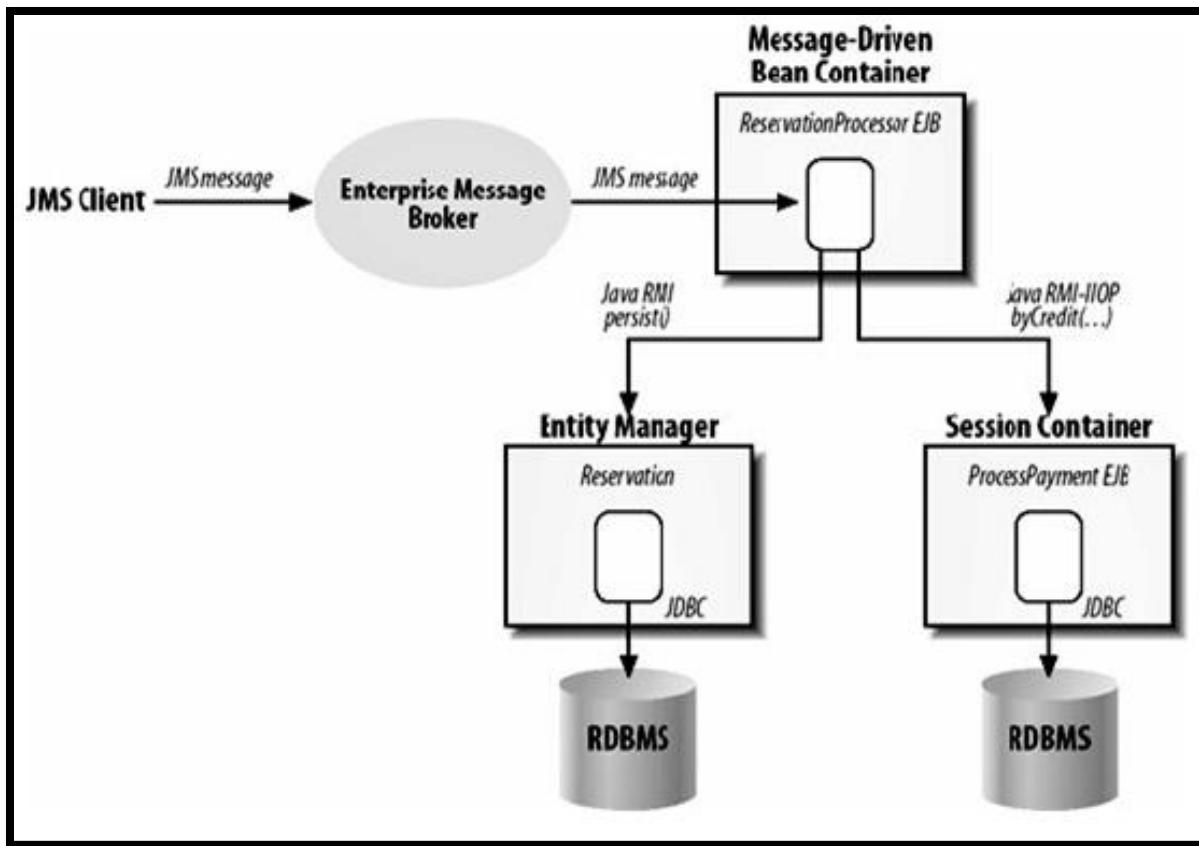


MDB Características

- ▶ Reciben un único mensaje del cliente
- ▶ Son invocados asincrónicamente
- ▶ No representan datos compartidos, pero pueden acceder y actualizar datos
- ▶ Pueden participar en transacciones
- ▶ No tienen estado
- ▶ El contenedor no garantiza el orden de procesamiento de los mensajes enviados a los MDB
 - ▶ Debido a la concurrencia

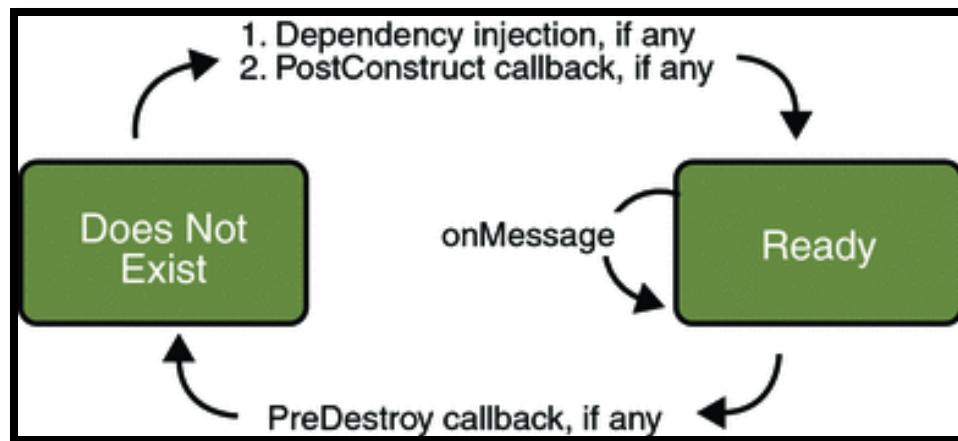
MDB ciclo de vida

- ▶ Se puede invocar un EJB para procesar ó guardar la información.



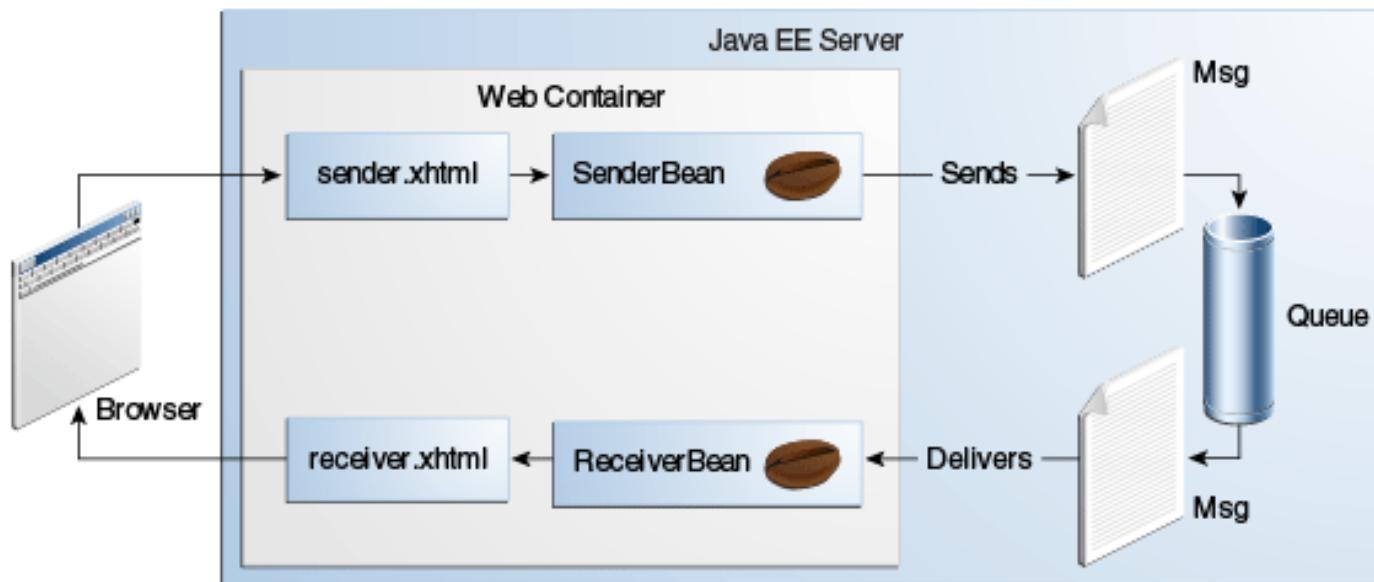
MDB ciclo de vida

- ▶ Cuando un mensaje llega el contenedor invoca el método **onmessage** de un MDB para procesar dicho mensaje



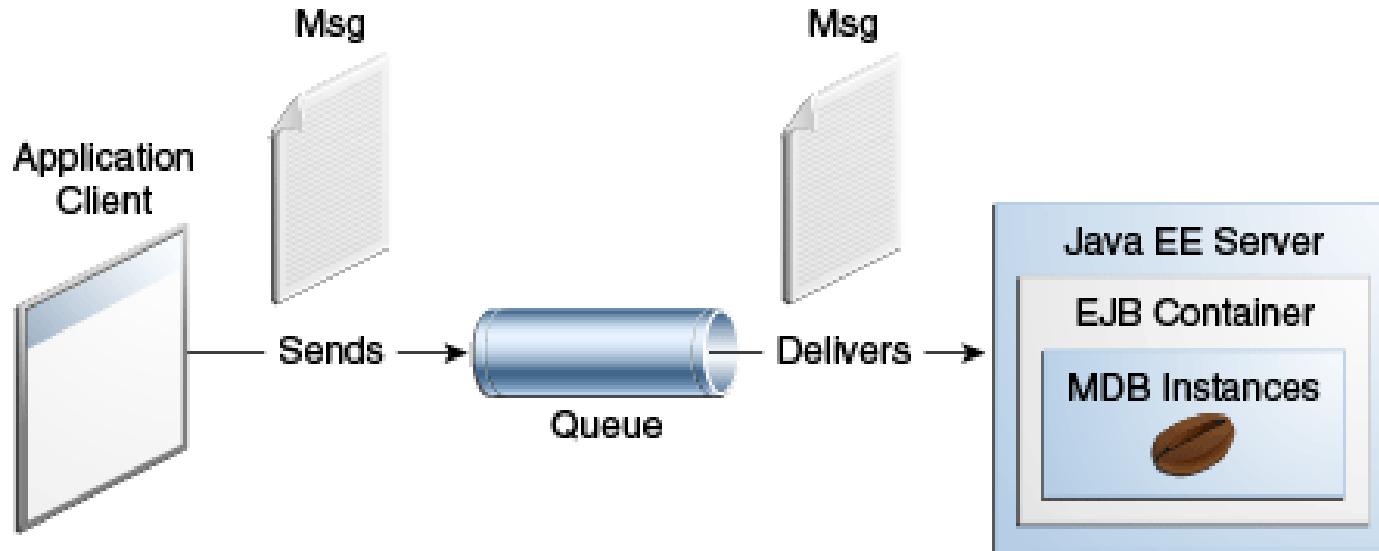
Ejemplo consumo sincrónico

- ▶ Sender Bean: produce
- ▶ Receiver Bean: consume sincrónicamente



Ejemplo consumo asincrónico

- ▶ Application Client: produce el mensaje
- ▶ MDB: consume el mensaje de manera asincrónica

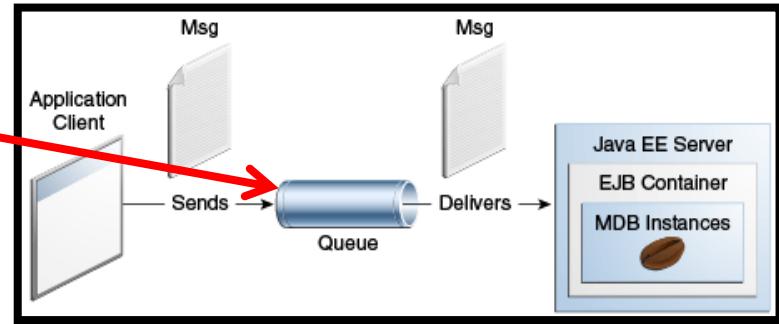


Ejemplo consumo asincrónico

```
@Resource(lookup = "java:comp/DefaultJMSConnectionFactory")  
private static ConnectionFactory connectionFactory;
```

JNDI

```
@Resource(lookup = "jms/MyQueue")  
private static Queue queue;  
String text;  
final int NUM_MSGS = 3;
```



```
try (JMSContext context = connectionFactory.createContext()) {
```

Finally, the client sends several text messages to the queue:

```
for (int i = 0; i < NUM_MSGS; i++) {  
    text = "This is message " + (i + 1);  
    System.out.println("Sending message: " + text);  
    context.createProducer().send(queue, text);  
}
```

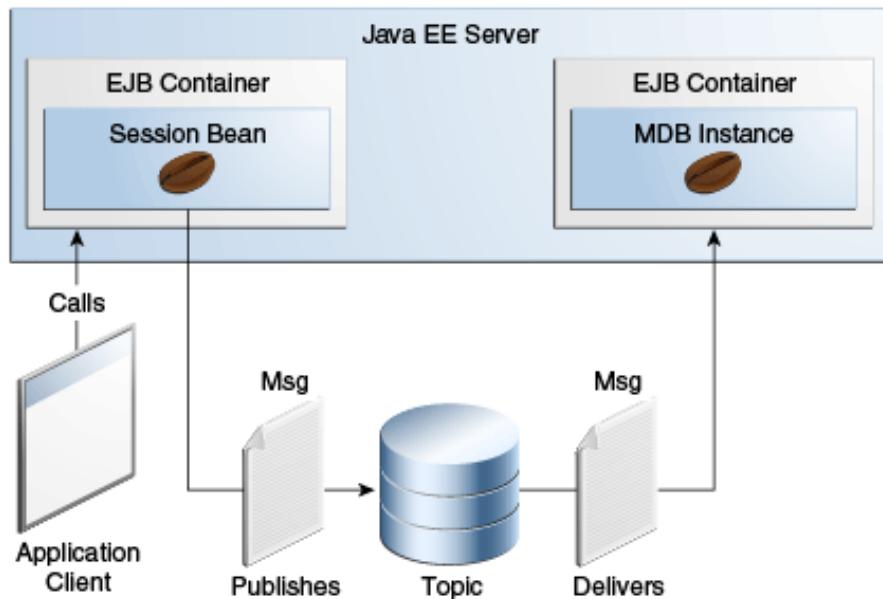
Ejemplo: Consumidor MDB

```
@MessageDriven(activationConfig = {  
    @ActivationConfigProperty(propertyName = "destinationLookup",  
        propertyValue = "jms/MyQueue"),  
    @ActivationConfigProperty(propertyName = "destinationType",  
        propertyValue = "javax.jms.Queue")  
})  
  
public class SimpleMessageBean implements MessageListener {  
  
    @Resource  
    private MessageDrivenContext mdc;  
    static final Logger logger = Logger.getLogger("SimpleMessageBean");  
  
    public SimpleMessageBean() {  
    }  
  
    @Override  
    public void onMessage(Message inMessage) {  
  
        try {  
            if (inMessage instanceof TextMessage) {  
                logger.log(Level.INFO,  
                    "MESSAGE BEAN: Message received: {0}",  
                    inMessage.getBody(String.class));  
            } else {  
                logger.log(Level.WARNING,  
                    "Message of wrong type: {0}",  
                    inMessage.getClass().getName());  
            }  
        } catch (JMSException e) {  
            logger.log(Level.SEVERE,  
                "SimpleMessageBean.onMessage: JMSException: {0}",  
                e.toString());  
            mdc.setRollbackOnly();  
        }  
    }  
}
```

Los message listener manejan la recepción asíncrona de mensajes implementando el onMessage method

Ejemplo 3: EJB SBSL produce a un Topic

- ▶ Session Bean: produce
- ▶ MDB: consume



```
@Stateless  
@Remote({  
    PublisherRemote.class  
})  
public class PublisherBean implements PublisherRemote {  
  
    @Resource  
    private SessionContext sc;  
    @Resource(lookup = "java:module/jms/newsTopic")  
    private Topic topic;  
    @Inject  
    private JMSContext context;  
    ...}
```

Mensajes en JMS

- ▶ Un mensaje está compuesto de:
 - ▶ Header
 - ▶ Properties
 - ▶ Body

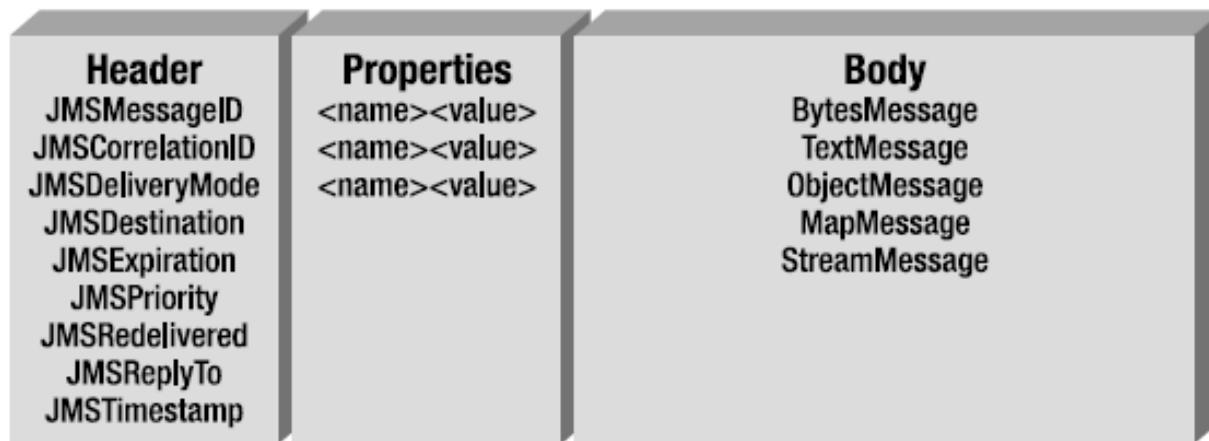


Figure 13-8. Structure of a JMS message

Cabecera del mensaje JMS

- `JMSDestination` is set through `send` or `publish`
- `JMSDeliveryMode` is set through `send` or `publish`
- `JMSExpiration` is set through `send` or `publish`
- `JMSPriority` is set through `send` or `publish`
- `JMSMessageID` is set through `send` or `publish`
- `JMSTimestamp` is set through `send` or `publish`
- `JMSCorrelationID` is set by the client
- `JMSReplyTo` is set by the client
- `JMSType` is set by the client
- `JMSRedelivered` is set by the JMS provider

Tipos del mensaje en el cuerpo

- TextMessage: a literal string or XML.
- MapMessage: key-value pairs similar to map. E.g.
`getInt("numberOfItems")`.
- BytesMessage: a stream of raw bytes.
- ObjectMessage: a single serializable object.
- StreamMessage: a stream of Java primitives like int.
- Message: an empty body.

Tutorial de Topic y MDB

- ▶ http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/JMS_pub_sub/JMS_pub_sub.html

Crear Recursos JMS desde NB

- ▶ <http://docs.oracle.com/javaee/7/tutorial/doc/jms-examples009.htm>

Bibliografía

- ▶ **Beginning Java EE 7.** Copyright © 2013 by Antonio Goncalves.



Bibliografia

<http://docs.oracle.com/javaee/7/tutorial/doc/home.htm>

Bibliografía

- ▶ The Java EE 6/7/8 Tutorial.

Bibliografía

- ▶ **Beginning Java EE 7.** Antonio Goncalves.
- ▶ Tutorial JavaEE 7:
<http://docs.oracle.com/javaee/7/tutorial/doc/home.htm>





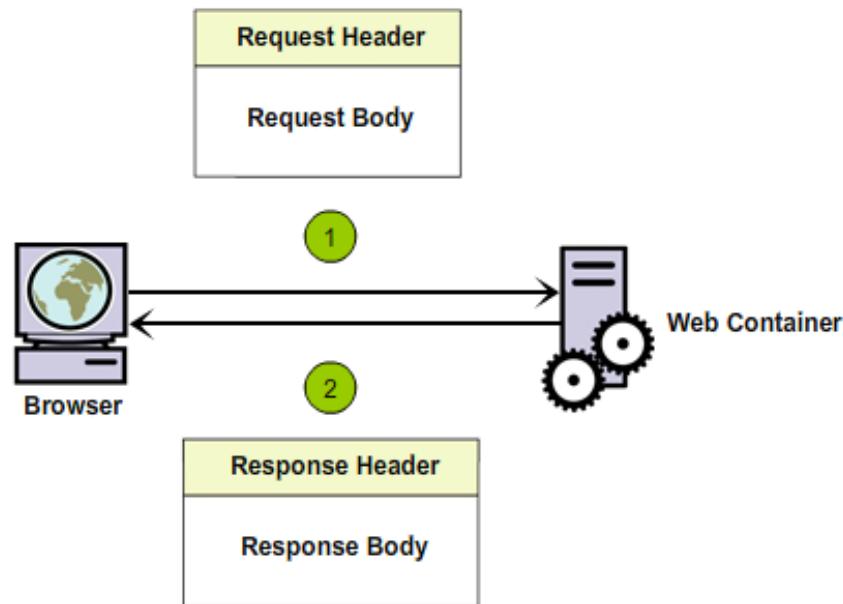
Arquitectura de Software



HTTP

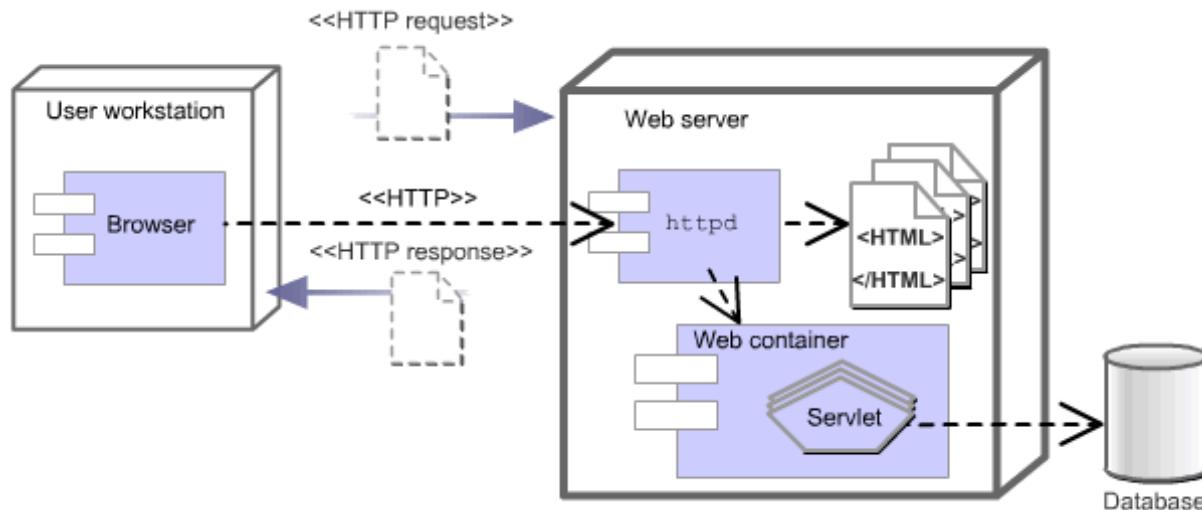
HTTP Modelo

- ▶ HTTP es un web estándar basadas en peticiones (request) y respuestas (response)



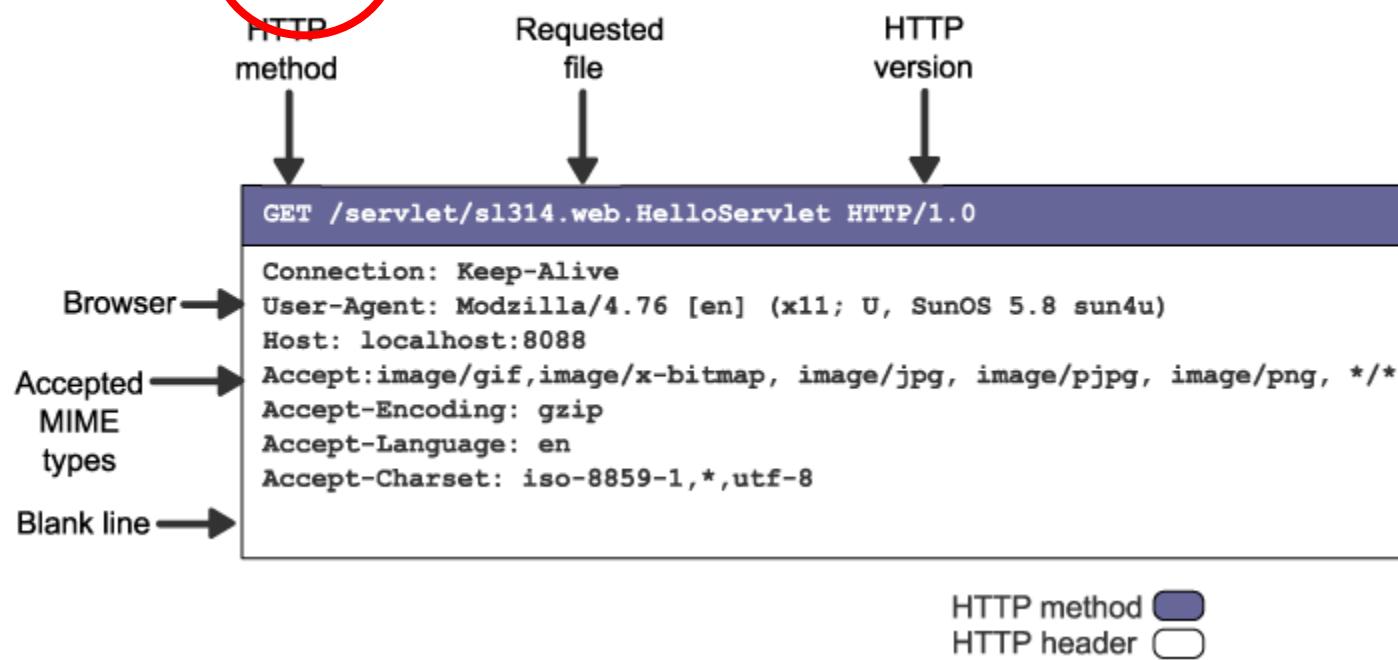
HTTP Modelo

- ▶ Ejemplos:
 - ▶ solicitar una pagina web (HTML)
 - ▶ Solicitar datos (JSON/XML)



HTTP Request

Example HTTP Request Stream



HTTP Methods

HTTP Method	Request has body	Response has body	Safe	Idempotent	Cachable
GET	NO	YES	YES	YES	YES
POST	YES	YES	NO	NO	YES
PUT	YES	YES	NO	YES	NO
DELETE	YES	YES	NO	YES	NO
TRACE	NO	YES	YES	YES	NO
OPTIONS	NO	YES	YES	YES	NO
CONNECT	NO	YES	NO	NO	NO
PATCH	YES	YES	NO	NO	NO



HTTP Response

Example HTTP Response Stream



HTTP Response Codes

HTTP Status Codes



Informational Status Codes

100 – Continue [The server is ready to receive the rest of the request.]

101 – Switching Protocols [Client specifies that the server should use a certain protocol and the server will give this response when it is ready to switch.]

Client Request Successful

200 – OK [Success! This is what you want.]

201 – Created [Successfully created the URL specified by the client.]

202 – Accepted [Accepted for processing but the server has not finished processing it.]

203 – Non-Authoritative Information [Information in the response header did not originate from this server. Copied from another server.]

204 – No Content [Request is complete without any information being sent back in the response.]

205 – Reset Content [Client should reset the current document. I.e. A form with existing values.]

206 – Partial Content [Server has fulfilled the partial GET request for the resource. In response to a Range request from the client. Or if someone hits stop.]

Request Redirected

300 – Multiple Choices [Requested resource corresponds to a set of documents. Server sends information about each one and a URL to request them from so that the client can choose.]

301 – Moved Permanently [Requested resource does not exist on the server. A Location header is sent to the client to redirect it to the new URL. Client continues to use the new URL in future requests.]

302 – Moved Temporarily [Requested resource has temporarily moved. A Location header is sent to the client to redirect it to the new URL. Client continues to use the old URL in future requests.]

303 – See Other [The requested resource can be found in a different location indicated by the Location header, and the client should use the GET method to retrieve it.]

304 – Not Modified [Used to respond to the If-Modified-Since request header. Indicates that the requested document has not been modified since the the specified date, and the client should use a cached copy.]

305 – Use Proxy [The client should use a proxy, specified by the Location header, to retrieve the URL.]

307 – Temporary Redirect [The requested resource has been temporarily redirected to a different location. A Location header is sent to redirect the client to the new URL. The client continues to use the old URL in future requests.]

Client Request Incomplete

400 – Bad Request [The server detected a syntax error in the client's request.]

401 – Unauthorized [The request requires user authentication. The server sends the WWW-Authenticate header to indicate the authentication type and realm for the requested resource.]

402 – Payment Required [reserved for future.]

403 – Forbidden [Access to the requested resource is forbidden. The request should not be repeated by the client.]

404 – Not Found [The requested document does not exist on the server.]

405 – Method Not Allowed [The request method used by the client is unacceptable. The server sends the Allow header stating what methods are acceptable to access the requested resource.]

406 – Not Acceptable [The requested resource is not available in a format that the client can accept, based on the accept headers received by the server. If the request was not a HEAD request, the server can send Content-Language, Content-Encoding and Content-Type headers to indicate which formats are available.]

407 – Proxy Authentication Required [Unauthorized access request to a proxy server. The client must first authenticate itself with the proxy. The server sends the Proxy-Authenticate header indicating the authentication scheme and realm for the requested resource.]

408 – Request Time-Out [The client has failed to complete its request within the request timeout period used by the server. However, the client can re-request.]

409 – Conflict [The client request conflicts with another request. The server can add information about the type of conflict along with the status code.]

410 – Gone [The requested resource is permanently gone from the server.]

411 – Length Required [The client must supply a Content-Length header in its request.]

412 – Precondition Failed [When a client sends a request with one or more If... headers, the server uses this code to indicate that one or more of the conditions specified in these headers is FALSE.]

413 – Request Entity Too Large [The server refuses to process the request because its message body is too large. The server can close connection to stop the client from continuing the request.]

414 – Request-URI Too Long [The server refuses to process the request, because the specified URI is too long.]

415 – Unsupported Media Type [The server refuses to process the request, because it does not support the message body's format.]

417 – Expectation Failed [The server failed to meet the requirements of the Expect request-header.]

Server Errors

500 – Internal Server Error [A server configuration setting or an external program has caused an error.]

501 – Not Implemented [The server does not support the functionality required to fulfill the request.]

502 – Bad Gateway [The server encountered an invalid response from an upstream server or proxy.]

503 – Service Unavailable [The service is temporarily unavailable. The server can send a Retry-After header to indicate when the service may become available again.]

504 – Gateway Time-Out [The gateway or proxy has timed out.]

505 – HTTP Version Not Supported [The version of HTTP used by the client is not supported.]

Unused status codes

306- Switch Proxy

416- Requested range not satisfiable

506- Redirection failed



Web Services

REST



Intro

REST

► **REPRESENTATIONAL STATE TRANSFER**



REST

- ▶ The term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
- ▶ Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.

Key REST Concepts

- Rest consists of 4 key concepts:
 - Resources
 - Resource names (URIs)
 - Resource representations
 - Links between resources

Resources

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g., “today’s weather in Los Angeles”), a collection of other resources, a nonvirtual object (e.g., a person), a concept and so on.

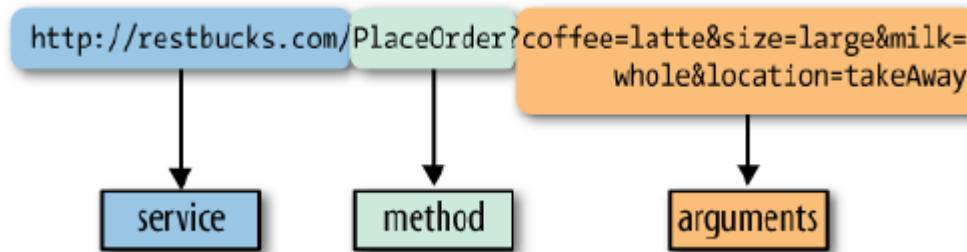
RESTFUL WEB SERVICE DESIGN GUIDELINE 1:

Every object manipulated by the web service (or web application) should be identified and exposed as a [resource](#).



URIs: names for resources

- A URI is the name of a resource
- Examples:
 - <http://www.lesoir.be>
 - <http://www.lesoir.be/edition/20-01-2012>
 - <http://www.example.org/newspapers/belgium>
 - <http://www.example.org/newspapers?country=belgium>



URIs: names for resources

RESTFUL WEB SERVICE DESIGN GUIDELINE 2:

- Every identified resource must be assigned at least one URI. This ensures it is **addressable**
- A URI should never represent more than one resource.
- Resources can have multiple URIs, but should have as few URIs as possible.

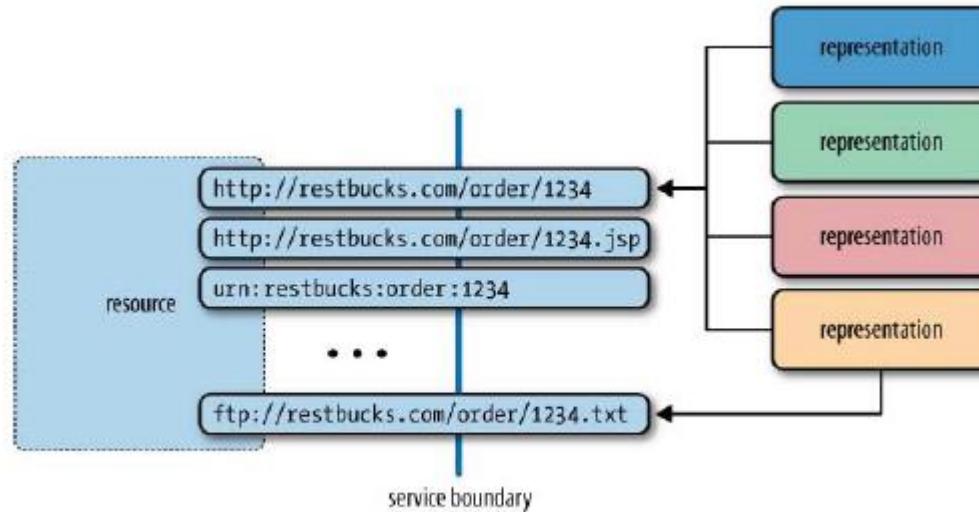
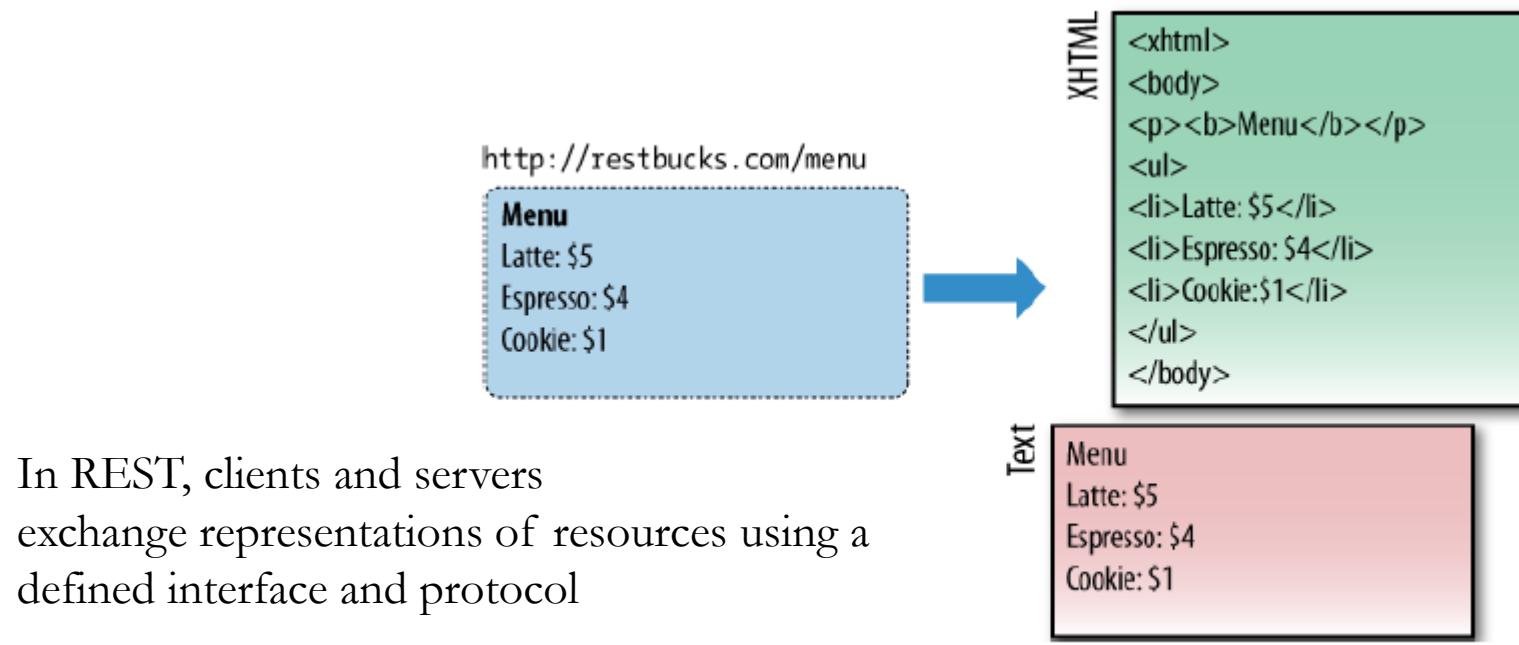


Figure 1-2. Multiple URIs for a resource

Representations

- A **representation** is a description of (some part of) the resource.
- A resource can have multiple representations (one in HTML, one in XML, one in a Google protocol buffer, ...).



In REST, clients and servers exchange representations of resources using a defined interface and protocol

Figure 1-3. Example of a resource and its representations

Representations

RESTFUL WEB SERVICE DESIGN GUIDELINE 3:

Specify, for every resource:

- The representations that the service serves to the client
- The representations that the service accepts from the client

Use standard representations whenever possible

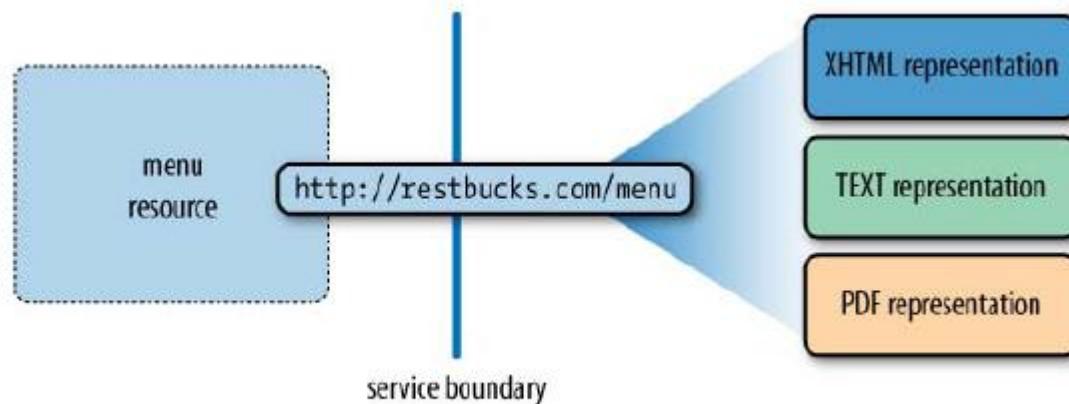


Figure 1-5. Multiple resource representations addressed by a single URI

The Uniform Interface

- All access to resources happens through HTTP uniform interface (GET, POST, PUT, DELETE, HEAD, OPTIONS).

CRUD	REST	
CREATE	POST	Create a (sub)resource
RETRIEVE	GET	Retrieve a representation of a resource
UPDATE	PUT	Modify a resource/create a new resource
DELETE	DELETE	delete a resource
	OPTIONS	Discover what HTTP methods are supported by the resource
	HEAD	requests headers only (similar to GET but omits representation)



The Uniform Interface (cont.)

RESOURCE RETRIEVAL

HTTP Client



Web Server



Database



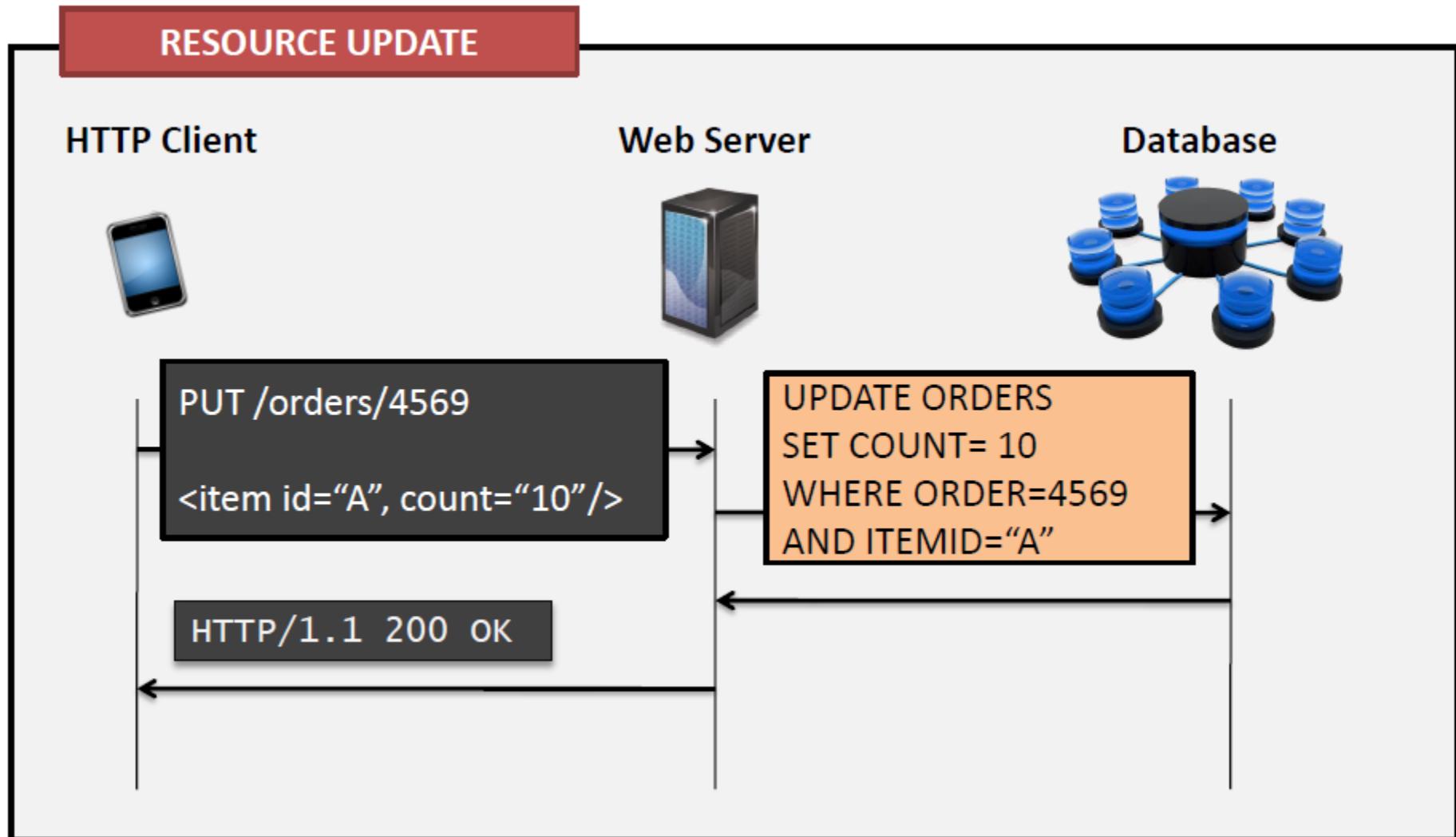
GET /books?isbn=122

SELECT * FROM BOOKS
WHERE ISBN=122

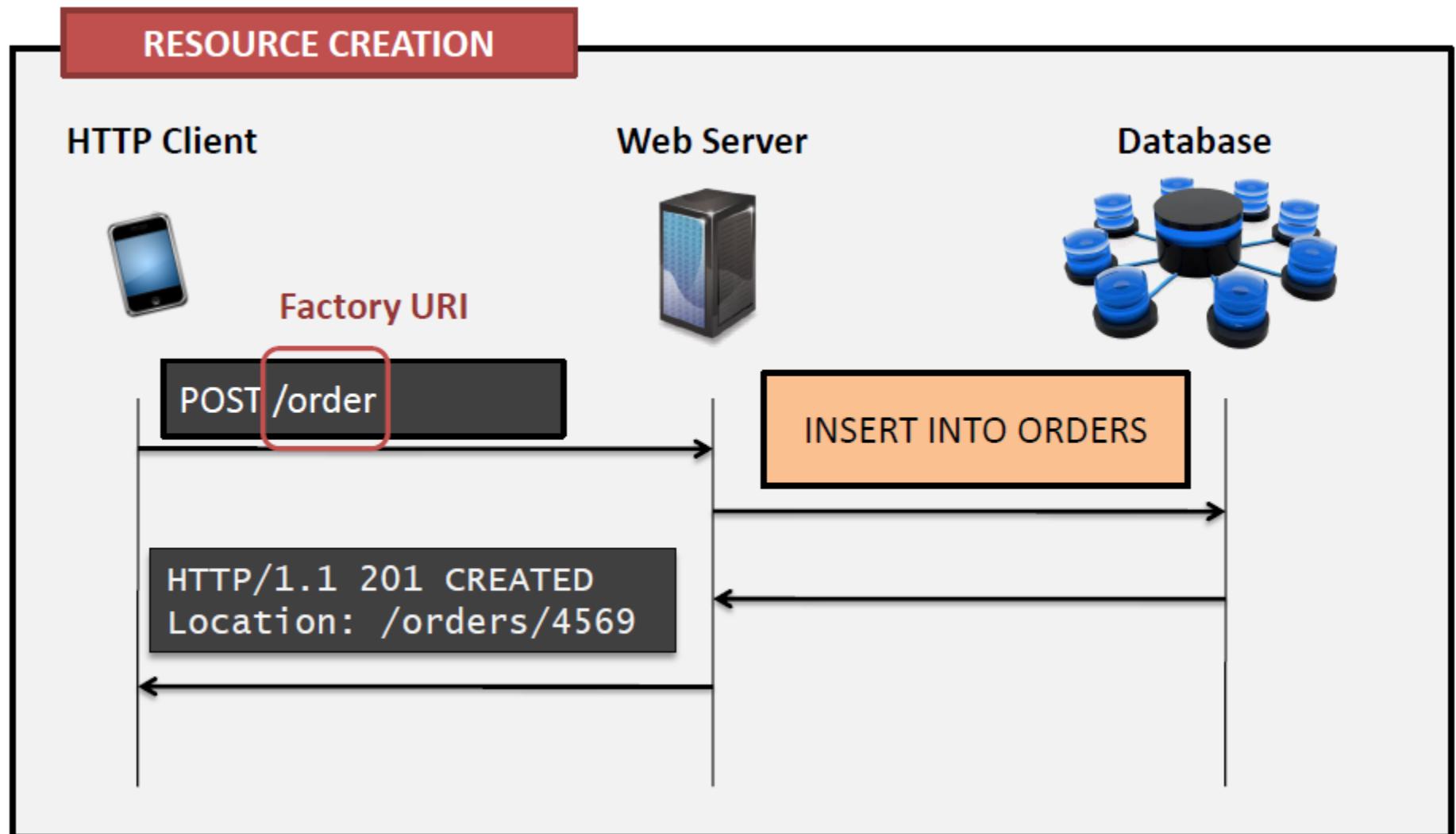
HTTP/1.1 200 OK

```
{ "title": "REST",  
  "authors":  
    ["Auth1", "Auth2"]  
}
```

The Uniform Interface (cont)



The Uniform Interface (cont)



The Uniform Interface (cont)

RESOURCE DELETION

HTTP Client



DELETE /orders/4569

HTTP/1.1 200 OK

GET /orders/4569

HTTP/1.1 404 NOT FOUND

Web Server



DELETE FROM ORDERS
WHERE ORDER=4569

Database



The Uniform Interface (cont)

- All access to resources happens through HTTP uniform interface (GET, POST, PUT, DELETE, HEAD, OPTIONS).
- All information necessary to understand the request must be contained in the request message.

RESTFUL WEB SERVICE DESIGN GUIDELINE 4:

Specify, for every URI (and hence, resource): the HTTP methods supported (e.g., GET and POST, but not DELETE, PUT)

Allow querying of these operations by supporting OPTIONS



The Uniform Interface (cont)

The de facto web protocol is HTTP, which is a document-based standardized request/response protocol between a client and a server. **HTTP is the uniform interface of RESTful web services.** Web services built on SOAP, WSDL, and other WS-* standards also use HTTP as the transport layer, but they leverage only a very few of its capabilities. You have to discover the semantic of the service by analyzing the WSDL and then invoke the right methods. RESTful web services have a uniform interface (HTTP methods and URIs), so, once you know where the resource is (URI), you can invoke the HTTP method (GET, POST, etc.).

In addition to familiarity, a uniform interface promotes interoperability between applications; HTTP is widely supported, and the number of HTTP client libraries guarantees that you won't have to deal with communication issues.



Why the Uniform Interface matters

- Consider a GET of
<http://api.del.icio.us/posts/delete>
- This misuses GET and does not adhere to the uniform interface
- But software programs don't know this. Programs that follow a link by GETTing it may hence (inadvertly) delete data. [e.g., Google Web Accelerator]

RESTFUL WEB SERVICE DESIGN GUIDELINE 5:

Use the HTTP methods correctly when designing web services.

On content negotiation and URIs

Content Negotiation

Content negotiation, described in section 12 of the HTTP standard, is defined as “the process of selecting the best representation for a given response when there are multiple representations available.” Clients’ needs, desires, and capabilities vary; the best representation for a mobile-device user in Japan might not be the best for a feed-reader application in the US.

Content negotiation is based on, but not limited to, the HTTP request headers `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`, and `User-Agent`. For example, to get the CSV representation of Apress Java books, the client application (the user agent) will request `http://www.apress.com/book/catalog/java` with a header `Accept` set to `text/csv`. You could also imagine that, based on the `Accept-Language` header, the server selects the proper CSV document to match the corresponding language (Japanese or English).

Content Types

HTTP uses Internet media types (originally called MIME types) in the `Content-Type` and `Accept` header fields in order to provide open and extensible data typing and type negotiation. Internet media types are divided into five discrete, top-level categories: `text`, `image`, `audio`, `video`, and `application`. These types are further divided into several subtypes (`text/plain`, `text/xml`, `text/xhtml`, etc.).



On content negotiation and URIs

- Content negotiation can be used to get distinct representations of the same resource.
 - <http://www.example.org/newspapers/belgium.json>
 - <http://www.example.org/newspapers/belgium.xml>



Statelessness

- Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any client state stored on the server.
- Application state is therefore kept entirely on the client.
- Resource state is of course still kept on the server



Statelessness (cont)

- Statelessness:
 - Improves reliability because it makes it easier to recover from partial failures
 - Improves scalability because:
 - Servers can quickly free computing resources after each request
 - Different requests can be handled by different servers (load balancing) exactly because the server doesn't have to manage resource usage across requests.



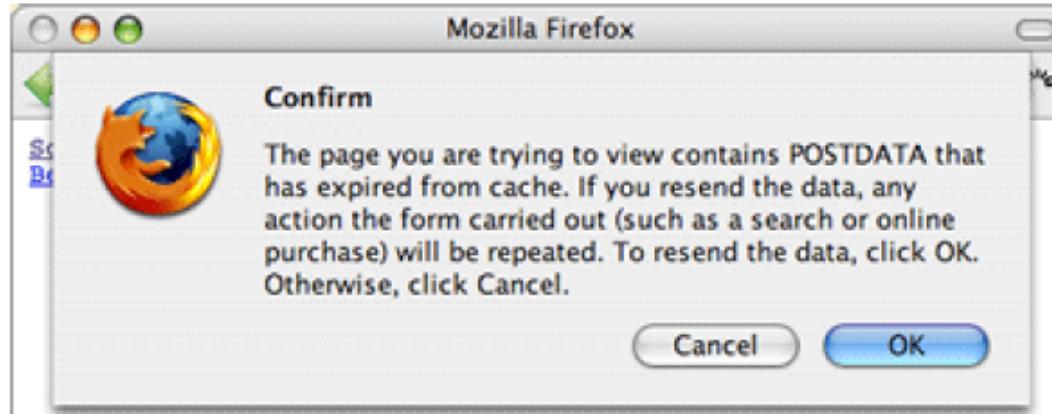
Statelessness (cont)

- Statelessness comes with many advantages such as better scalability: no session information to handle, no need to route subsequent requests to the same server, failure handling, and so on.



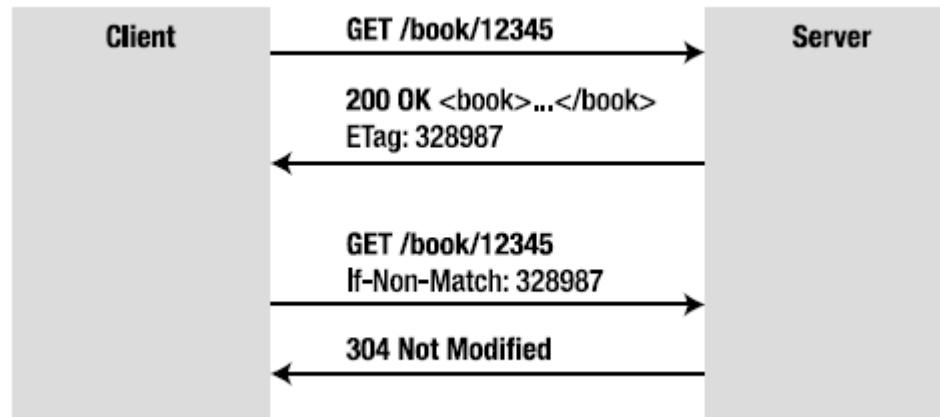
Safety & idempotency

- GET, HEAD, OPTIONS are read-only operations but PUT, POST, DELETE are read-write operations with side effects.
- An operation f is called **idempotent** if
$$f(f(x)) = f(x)$$
- PUT and DELETE are idempotent.
- Idempotent and read-only operations can safely be re-executed multiple times (e.g., network timeouts) without risking errors
- POST is not idempotent nor read-only, and is not safe to re-execute



Cache Representations

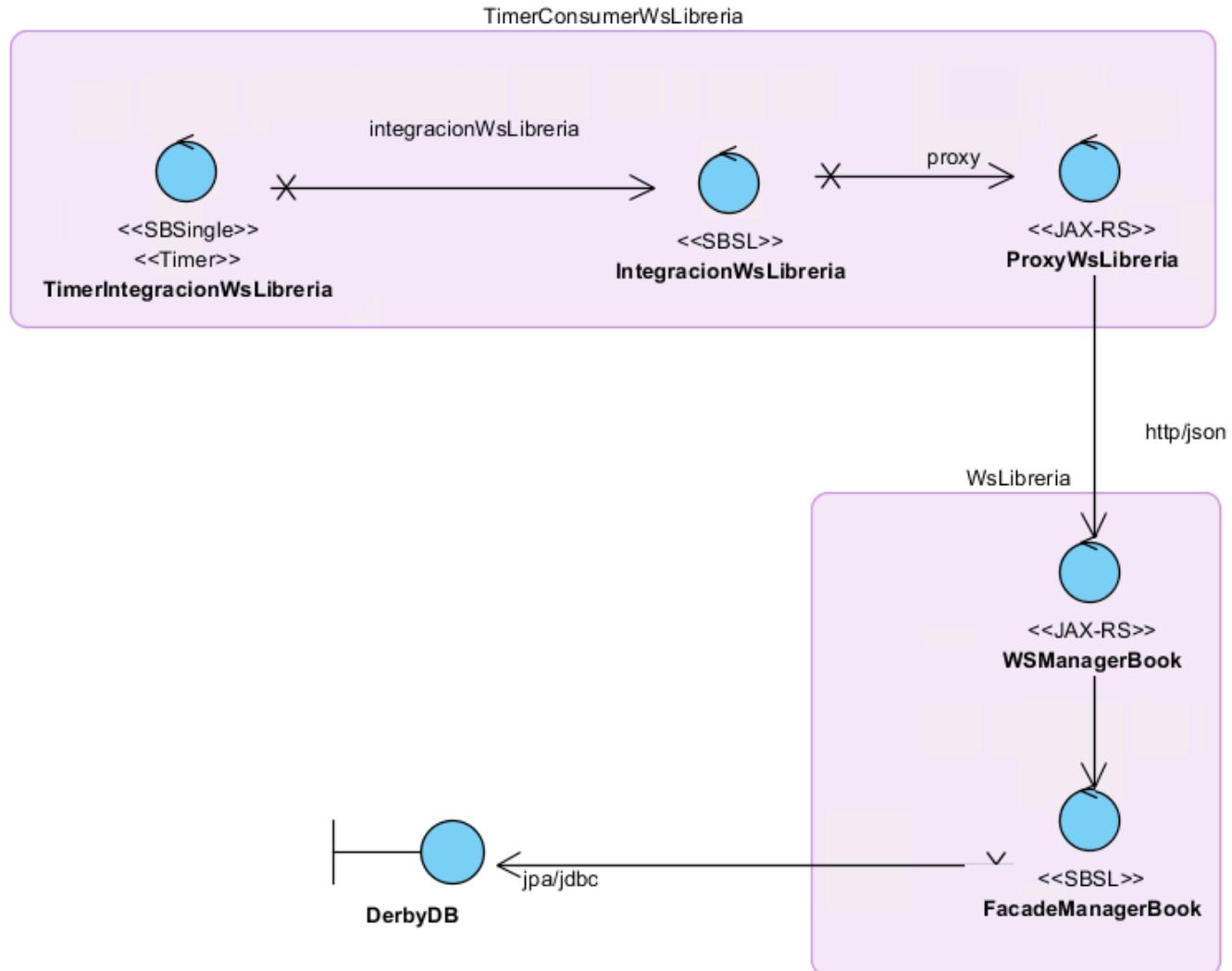
- In order to allow clients to cache representations that do not change frequently, the server should include the following headers:
 - Last-Modified
 - Etags
- This allows clients to use conditional get (using e.g., If-Modified-Since and If-None-Matches)



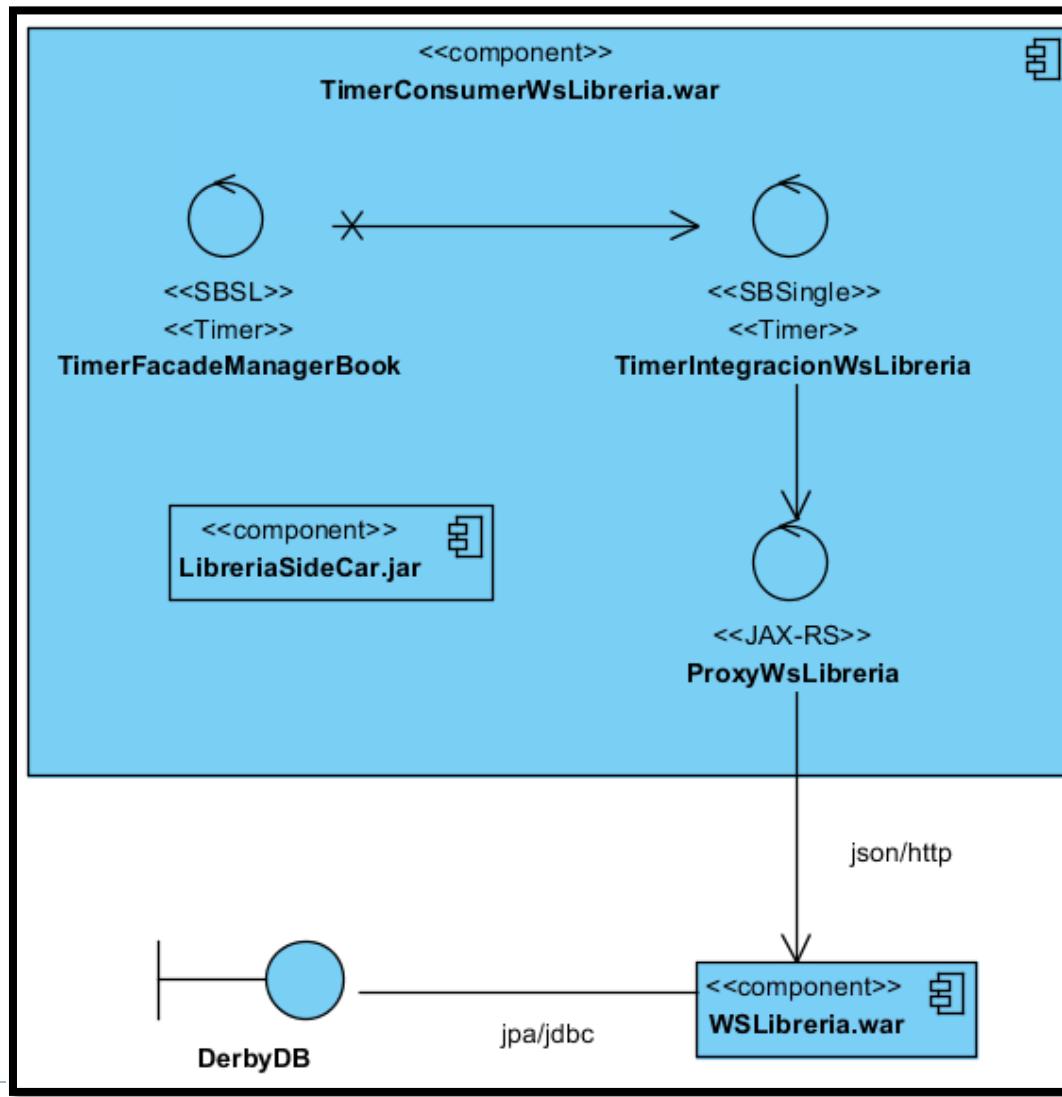
Arquitectura de Software Consumidor SBSL para REST JAX RS



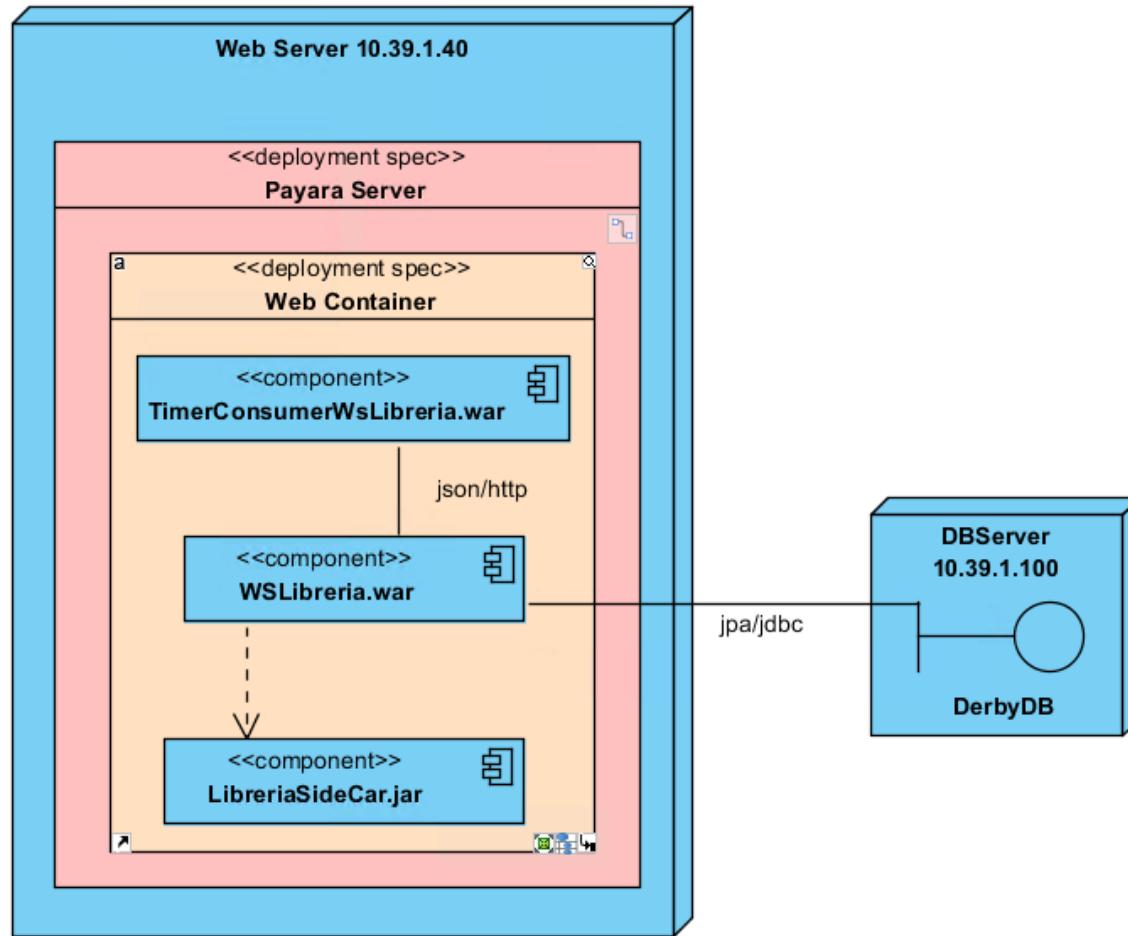
Arquitectura



Arquitectura



Arquitectura

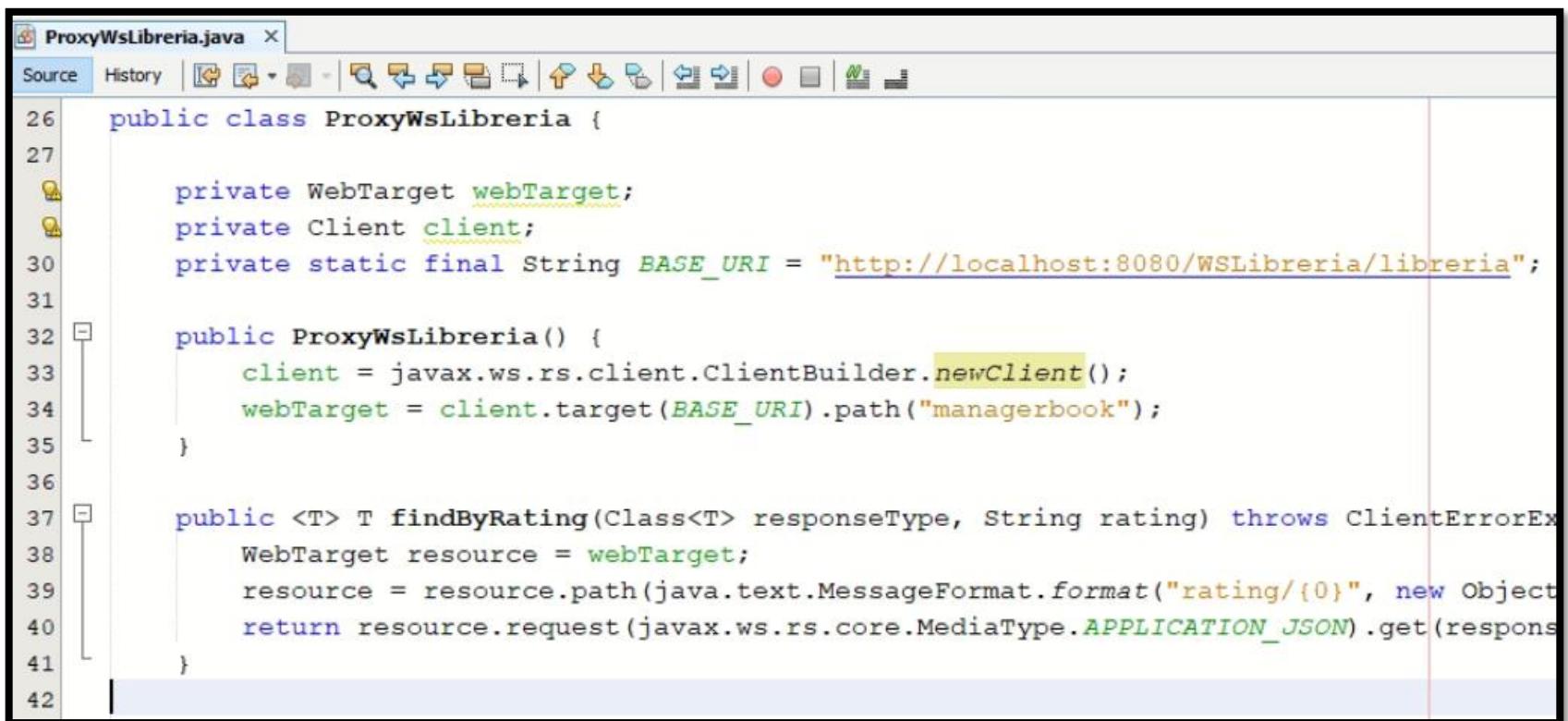


Inicio

- ▶ Las
funcionalidades
del servicio son=>

Proxy para el servicio de Libreria

Observe el archivo creado; se ve la URL del servicio (línea 30) y el Path (línea 34) del Rest (“managerbook”)

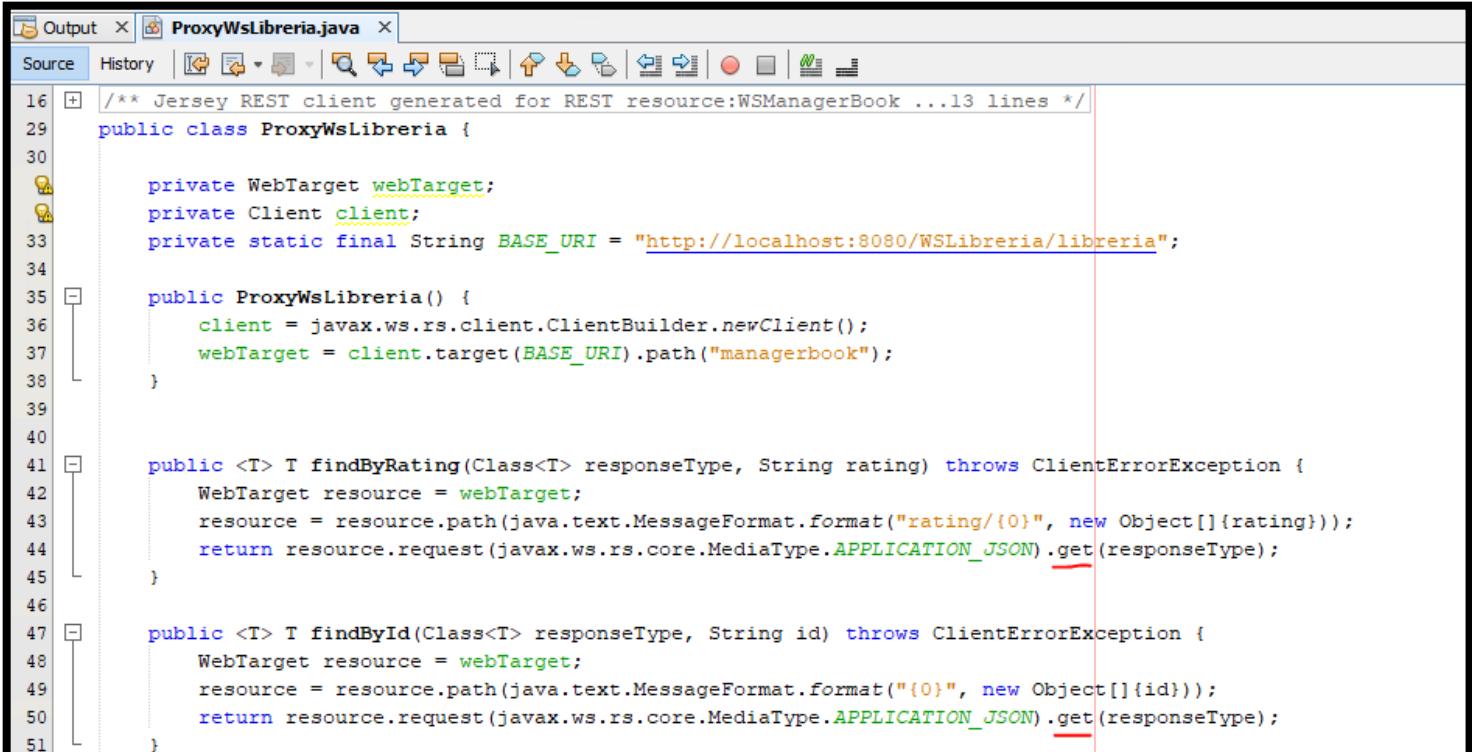


The screenshot shows a Java code editor with the file `ProxyWsLibreria.java` open. The code defines a class `ProxyWsLibreria` that interacts with a REST service. It includes fields for a `WebTarget` and a `Client`, and a static final string `BASE_URI` set to `"http://localhost:8080/WSLibreria/libreria"`. The constructor initializes the client and target, and a method `findByRating` performs a GET request to the service.

```
ProxyWsLibreria.java
Source History | 
26 public class ProxyWsLibreria {
27
28     private WebTarget webTarget;
29     private Client client;
30     private static final String BASE_URI = "http://localhost:8080/WSLibreria/libreria";
31
32     public ProxyWsLibreria() {
33         client = javax.ws.rs.client.ClientBuilder.newClient();
34         webTarget = client.target(BASE_URI).path("managerbook");
35     }
36
37     public <T> T findByRating(Class<T> responseType, String rating) throws ClientErrorException {
38         WebTarget resource = webTarget;
39         resource = resource.path(java.text.MessageFormat.format("rating/{0}", new Object[]
40             {rating}));
41         return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
42     }
}
```

Proxy para el servicio de Libreria

El proxy trae una invocación a cada método del servicio; el proxy invoca el servicio (en los diferentes métodos)y transforma el json a objetos JPA Book



The screenshot shows a Java code editor window titled "ProxyWsLibreria.java". The code is generated by Jersey REST client for a WSManagerBook resource. It defines a class "ProxyWsLibreria" with two methods: "findByRating" and "findById". Both methods use a "WebTarget" object to invoke a REST service at "http://localhost:8080/WSLibreria/libreria" and return objects of type T.

```
16  /** Jersey REST client generated for REST resource:WSManagerBook ...13 lines */
17
18  public class ProxyWsLibreria {
19
20      private WebTarget webTarget;
21      private Client client;
22      private static final String BASE_URI = "http://localhost:8080/WSLibreria/libreria";
23
24
25      public ProxyWsLibreria() {
26          client = javax.ws.rs.client.ClientBuilder.newClient();
27          webTarget = client.target(BASE_URI).path("managerbook");
28      }
29
30
31
32
33
34
35      public <T> T findByRating(Class<T> responseType, String rating) throws ClientErrorException {
36          WebTarget resource = webTarget;
37          resource = resource.path(java.text.MessageFormat.format("rating/{0}", new Object[]{rating}));
38          return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
39      }
40
41
42
43
44
45
46
47      public <T> T findById(Class<T> responseType, String id) throws ClientErrorException {
48          WebTarget resource = webTarget;
49          resource = resource.path(java.text.MessageFormat.format("{0}", new Object[]{id}));
50          return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
51      }
52
53
54
55
56
57
58
59
59 }
```

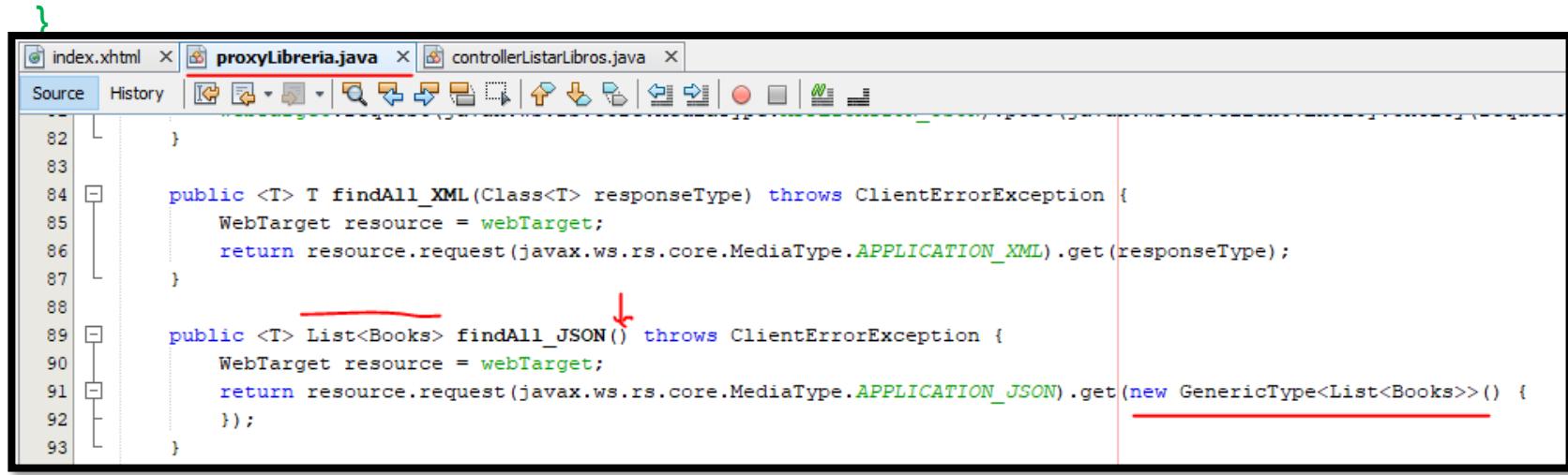
Proxy para el servicio de Libreria

El proxy trae una invocación a cada método del servicio; el proxy invoca el servicio (en los diferentes métodos)y transforma el json a objetos JPA Book

```
52
53     public Response addBook(Object requestEntity) throws ClientErrorException {
54         return webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).put(
55             javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON)
56             , Response.class);
57     }
58
59     public Response changeBook(Object requestEntity) throws ClientErrorException {
60         return webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).post(
61             javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON)
62             , Response.class);
63     }
64
65     public <T> T findAll(Class<T> responseType) throws ClientErrorException {
66         WebTarget resource = webTarget;
67         return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(responseType);
68     }
```

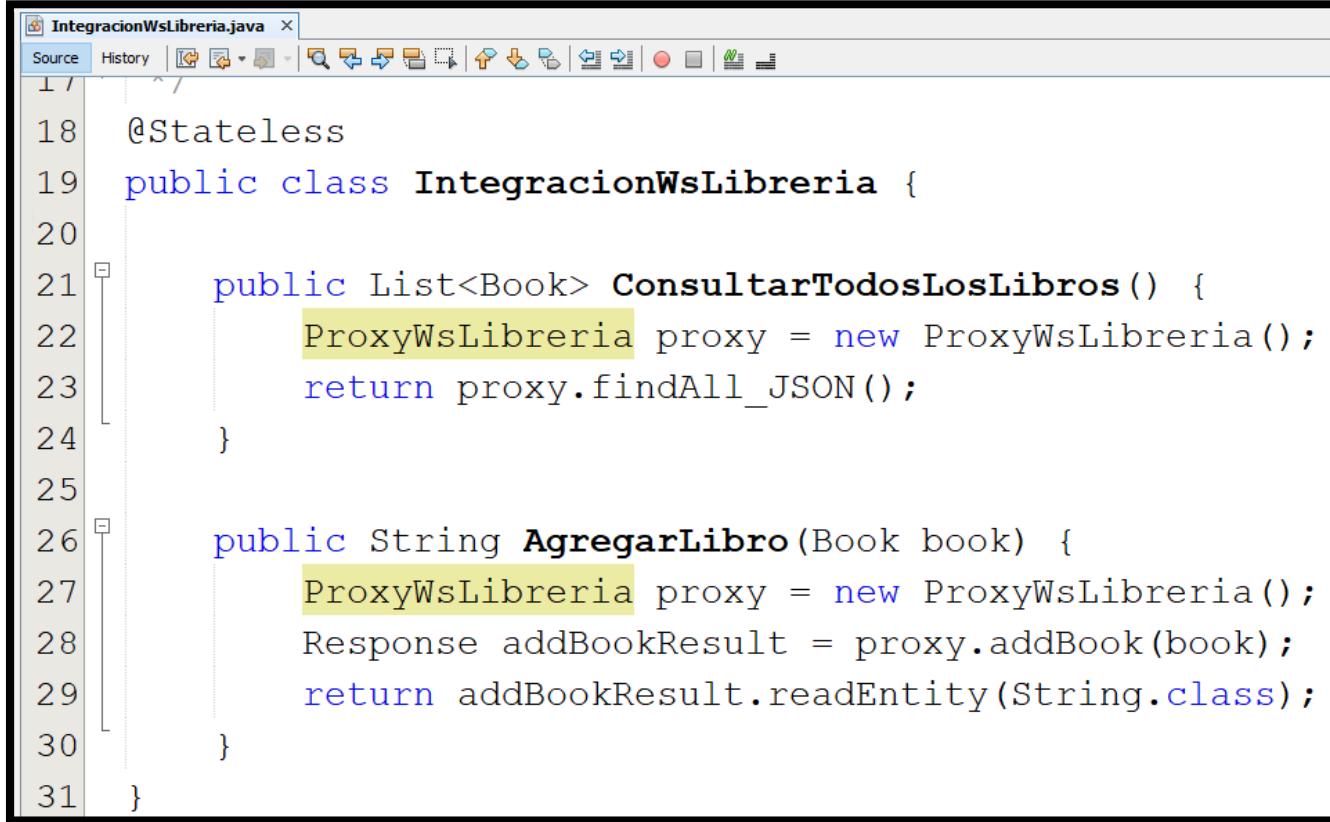
Proxy para el servicio de Libreria

```
public <T> List<Book> findAll_JSON() throws ClientErrorException {  
    WebTarget resource = webTarget;  
    return resource  
        .request(javax.ws.rs.core.MediaType.APPLICATION_JSON)  
        .get(new GenericType<List<Book>>() {  
    });  
}
```



```
index.xhtml x proxyLibreria.java x controllerListarLibros.java x  
Source History  
82     }  
83  
84     public <T> T findAll_XML(Class<T> responseType) throws ClientErrorException {  
85         WebTarget resource = webTarget;  
86         return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);  
87     }  
88  
89     public <T> List<Books> findAll_JSON() throws ClientErrorException {  
90         WebTarget resource = webTarget;  
91         return resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(new GenericType<List<Books>>());  
92     }  
93 }
```

IntegracionWsLibreria.java



```
17
18 @Stateless
19 public class IntegracionWsLibreria {
20
21     public List<Book> ConsultarTodosLosLibros() {
22         ProxyWsLibreria proxy = new ProxyWsLibreria();
23         return proxy.findAll_JSON();
24     }
25
26     public String AgregarLibro(Book book) {
27         ProxyWsLibreria proxy = new ProxyWsLibreria();
28         Response addBookResult = proxy.addBook(book);
29         return addBookResult.readEntity(String.class);
30     }
31 }
```

TimerIntegracionWsLibreria.java

```
16  /**...4 lines */
17
18  @Singleton
19
20  public class TimerIntegracionWsLibreria {
21
22      @EJB
23      IntegracionWsLibreria integridacionWsLibreria; 1
24
25      @Schedule(second = "*/2", minute = "*", hour = "*", persistent = false)
26
27      public void myTimer() {
28          System.out.println("Timer event: " + new Date());
29          List<Book> libros = integridacionWsLibreria.ConsultarTodosLosLibros(); 2
30          for (Book libro : libros) {
31              System.out.println("libro:" + libro.toString());
32          }
33      }
34
35  }
```



WADL

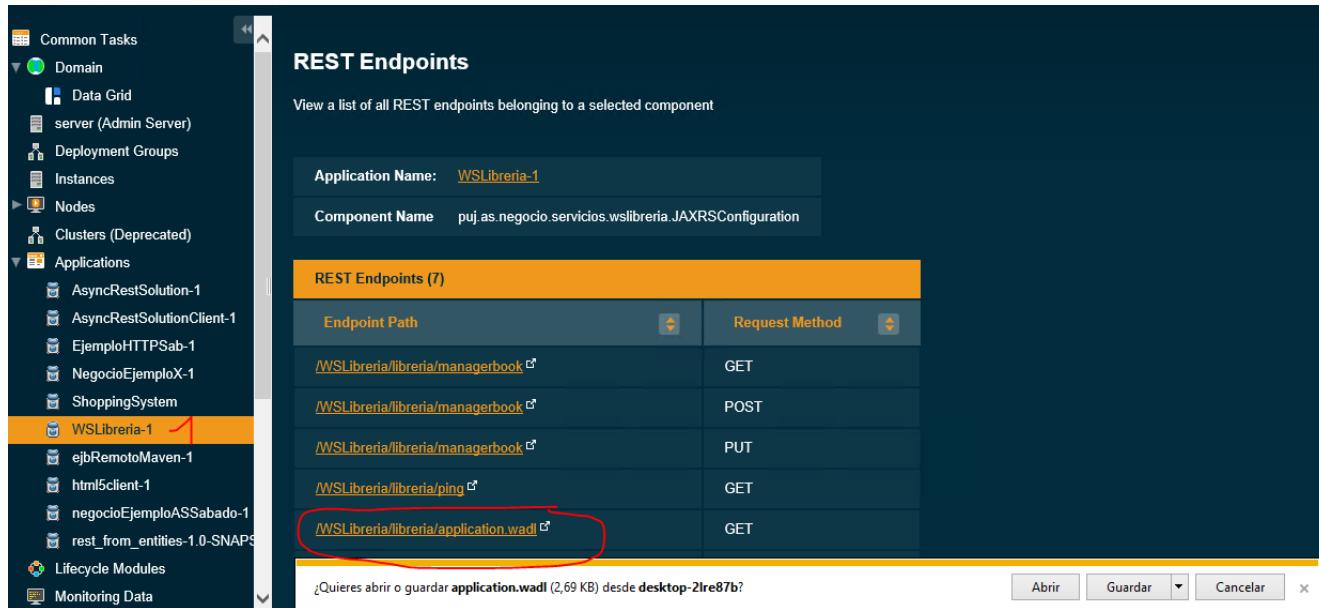
Usando el WADL

- ▶ ¿Como se crea un proxy desde un WADL?
- ▶ Esta opción se usa cuando nos envían el .wadl y no tenemos las fuentes del proyecto



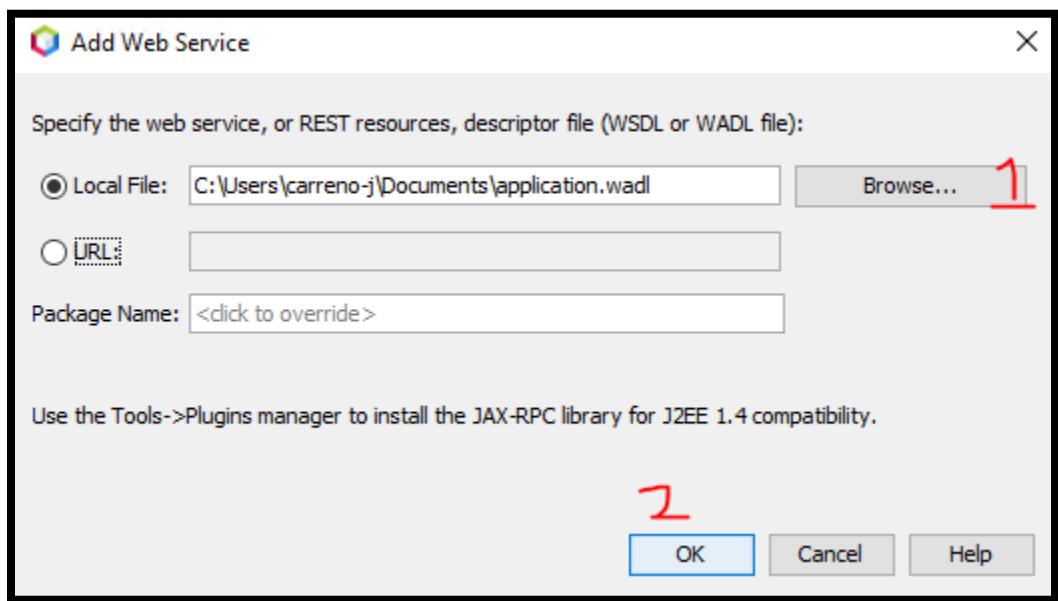
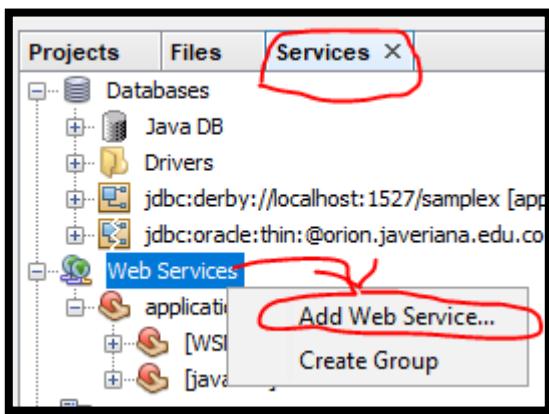
Usando el WADL para genera el proxy

- I. Obtenga el WADL; descárguelo a su disco duro



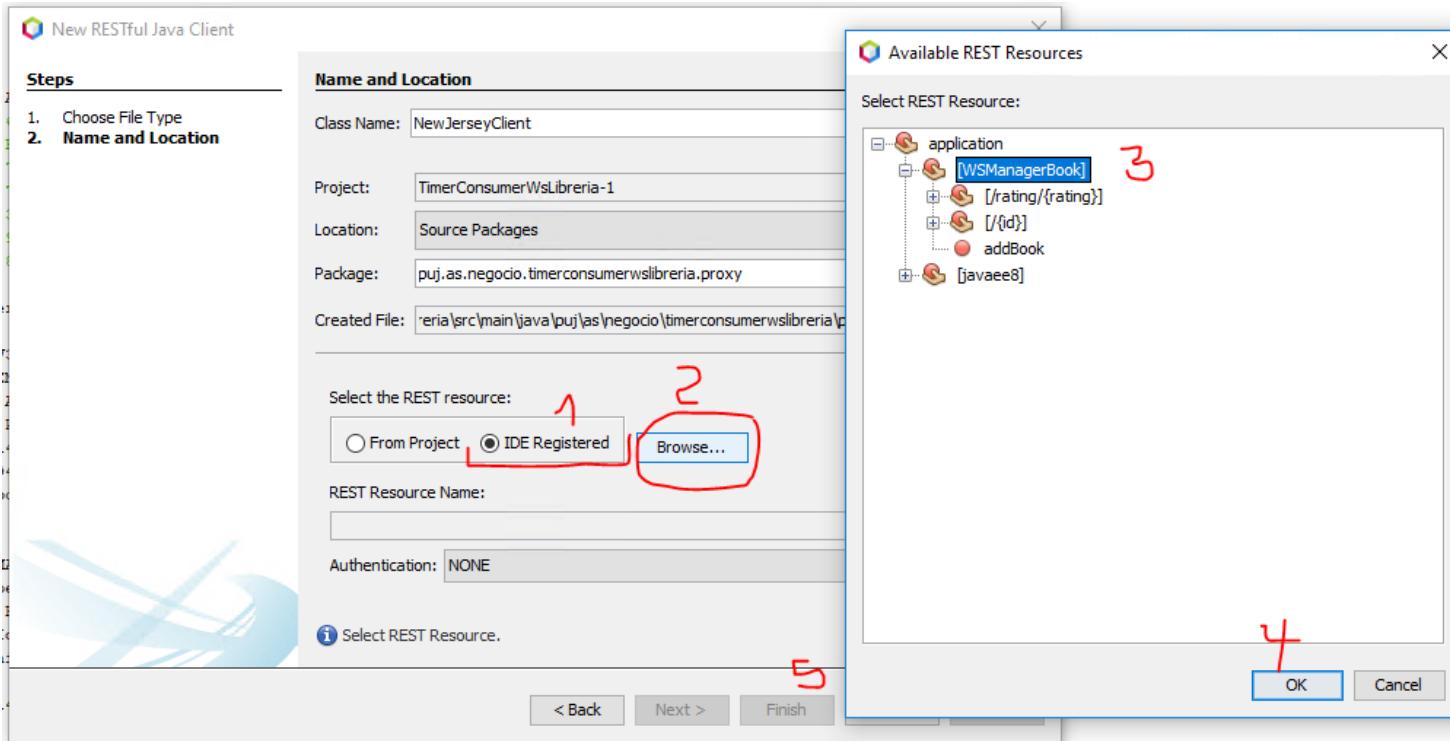
Usando el WADL para genera el proxy

► 2. Registre el WADL en el NB



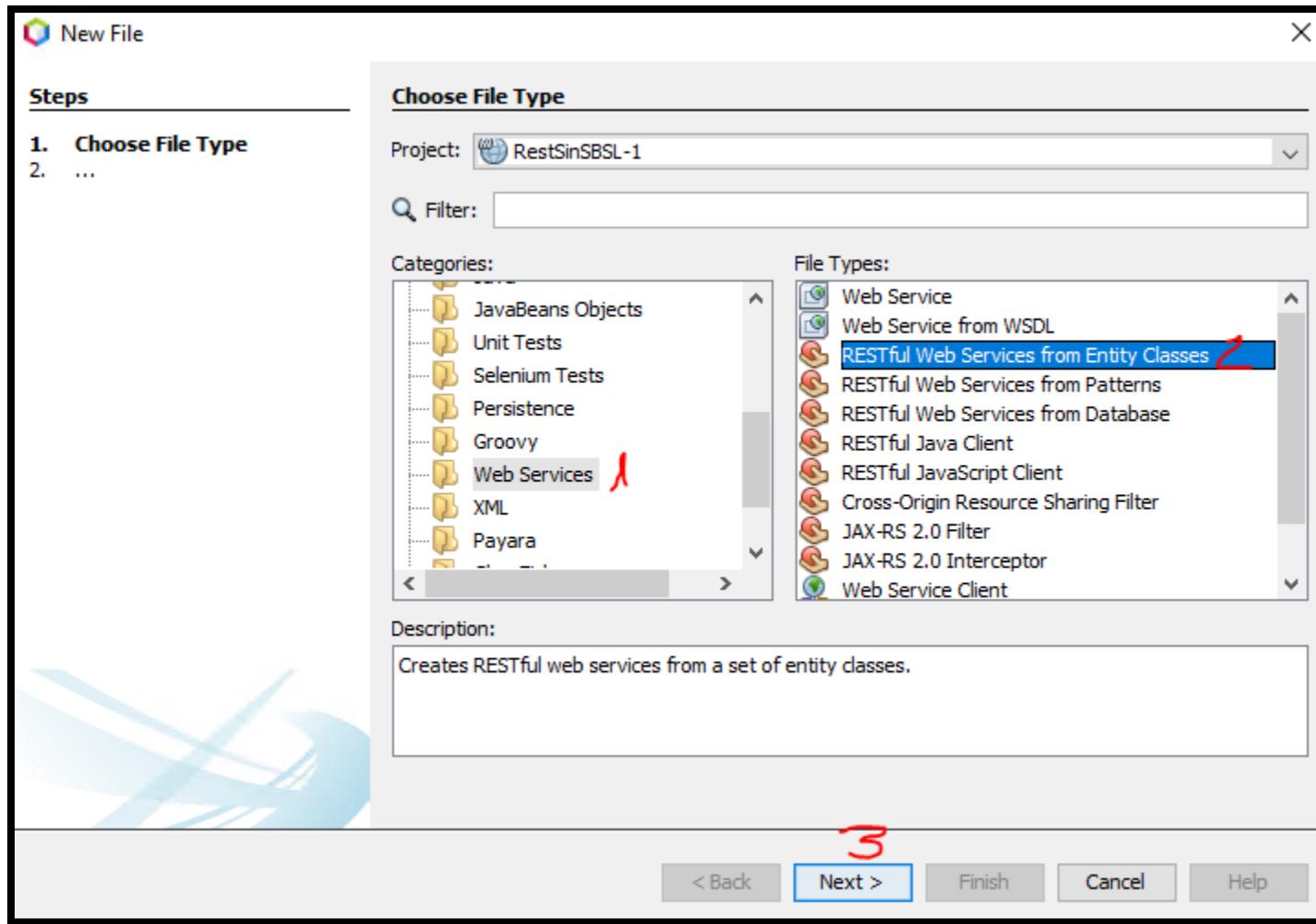
Usando el WADL para genera el proxy

► 3. use el WADL en el NB



JAX RS Rest Sin SBSL

Restful Sin SBSL



Restful Sin SBSL

The screenshot shows a Java code editor with the file `BookFacadeREST.java` open. The code implements a RESTful service for managing books. Annotations are highlighted with numbers:

- Annotation 1: `@Stateless` at line 28.
- Annotation 2: `@Path("puj.as.sidecarlibreria.entities.book")` at line 29.

```
25
26
27
28     @Stateless 1
29     @Path("puj.as.sidecarlibreria.entities.book") 2
30     public class BookFacadeREST extends AbstractFacade<Book> {
31
32         @PersistenceContext(unitName = "my_persistence_unit")
33         private EntityManager em;
34
35         public BookFacadeREST() {
36             super(Book.class);
37         }
38
39         @POST
40         @Override
41         @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
42         public void create(Book entity) {
43             super.create(entity);
44         }
45
46         @PUT
47         @Path("{id}")
48         @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
49         public void edit(@PathParam("id") String id, Book entity) {
50             super.edit(entity);
51         }
52
53     }
```



Otros Temas



Otros temas

- ▶ JAX RS (Restful) asincrónicos
- ▶ Restful Seguro
- ▶ Web services tipo SOAP
- ▶ SBSL Asincrónicos



References

- ▶ R. Daignau, Service Design Patterns, Addison-Wesley
- ▶ M. P. Papazoglou, Web Services: Principles and Technology, Prentice Hall

