Arquitectura de Aplicaciones

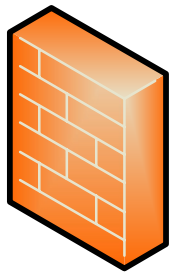
Sesión 1: Arquitectura de Aplicaciones

Puntos a tratar

- Arquitectura del Software
- Rol del Arquitecto
- Abstracción y Área de Superficie
- Arquitectura de 3 Capas: Capas vs Niveles
- Requisitos No Funcionales
- Arquitectura Ágil
- Roadmap

Introducción

- ¿Habéis visto a algún arquitecto levantar una pared antes de estudiar cual es la mejor orientación de una vivienda, estudiar si el suelo va a soportar 3 pisos en altura...?
  - Antes de construir cualquier casa, se necesita un proyecto de arquitectura donde se estudian y reflejan todos los elementos de la futura edificación.
- ¿ Similitud con el mundo "del ladrillo“ ?



- ¿Por qué cuando vamos a realizar una aplicación no realizamos primero un plano de la arquitectura del software?

Arquitectura y Software

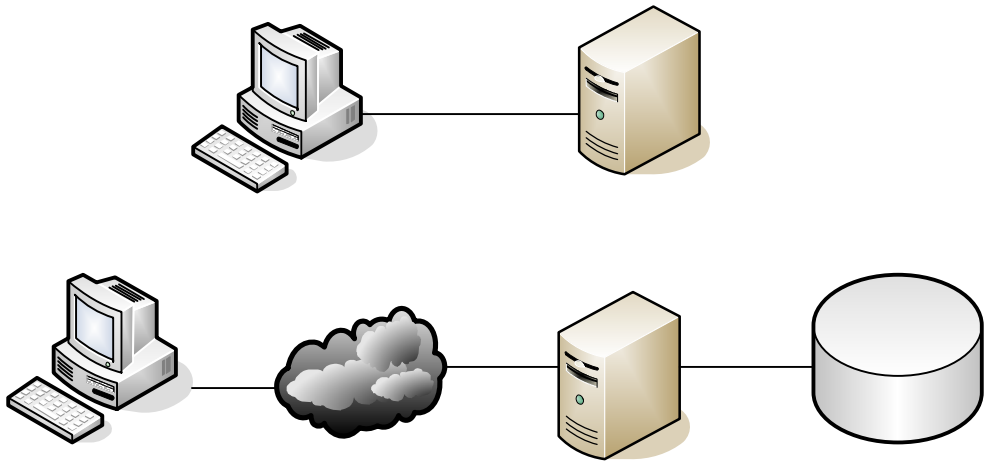
- Definiciones RAE de “**arquitectura**”:
  - Arte de proyectar y construir edificios
  - (*informática*) "Estructura lógica y física de los componentes de un computador".
- El concepto de *Arquitectura del Software* (AS)
  - nace a finales 60 de mano de *E. Dijkstra*
  - propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquiera manera.

Arquitectura del Software

- La **Arquitectura del Software** es la representación abstracta de los componentes y comportamientos de un sistema.
  - no contiene detalles de la implementación
  - el *arquitecto* trata con la información del problema y diseña una solución, la cual satisface los requerimientos funcionales y no funcionales del cliente, siendo una solución flexible que evoluciona cuando los requerimientos cambian.
- Es el esqueleto para construir un sistema.
- La arquitectura compone un conjunto de principios, estándares, protocolos, frameworks y directivas que dirigen los diversos elementos del diseño de toda aplicación.

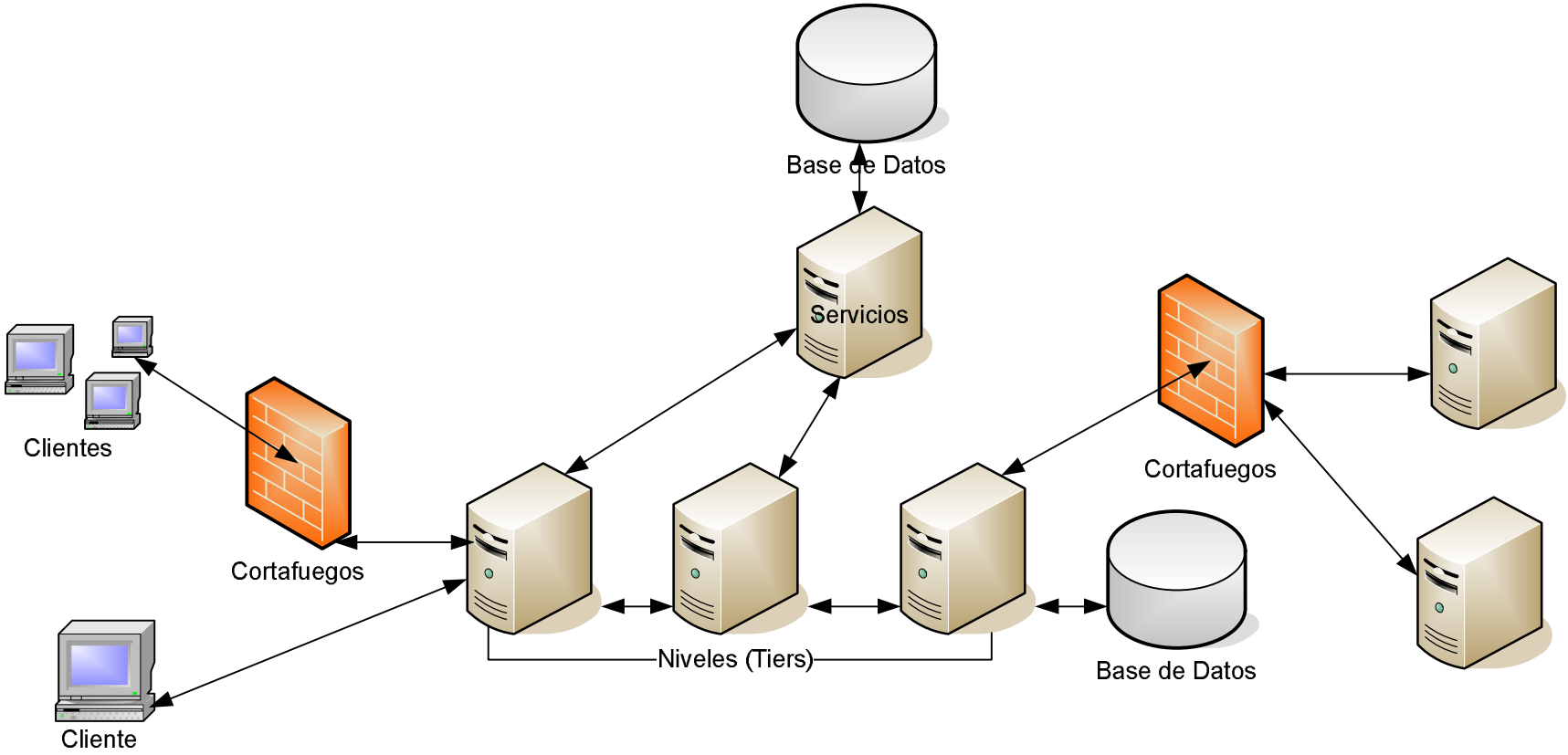
Necesidad

- ¿Por qué es tan importante ahora el concepto de arquitectura, si en el pasado no existía?
  - Intranet → Internet



Respuesta

- La respuesta esta en la **escalabilidad** y la **distribución** de las aplicaciones



AS como Proceso Creativo

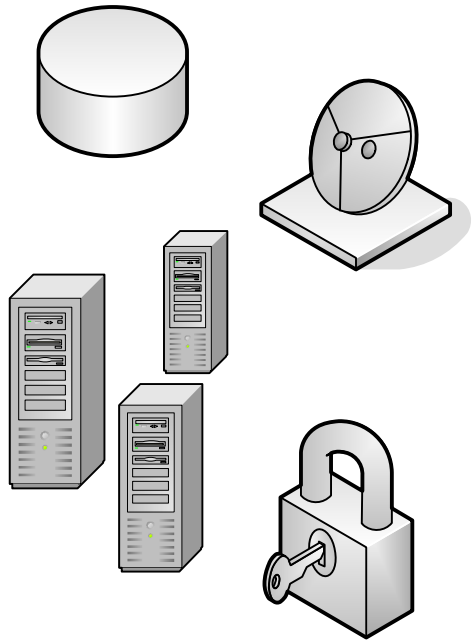
- Uno de los retos del arquitecto es equilibrar la creatividad con el pragmatismo
  - mediante las tecnologías disponibles en forma de modelos, frameworks y patrones.
- Puede hacer referencia a
  - un *producto* (la arquitectura de un edificio)
  - un método o *estilo* (la arquitectura de un rascacielos).
- Debe ser reconfigurable
  - para responder tanto a un entorno dinámico como a las peticiones por parte del cliente.

AS como Mecanismo

- Al crear una arquitectura describimos la estructura del sistema a construir, y como dicha estructura ofrece soporte a los requisitos de negocio y no funcionales.
- Un **mecanismo** es una capacidad que ofrece soporte a los requisitos de negocio de un modo consistente y uniforme.
  - Por ejemplo, la persistencia es una mecanismo que debería utilizarse consistentemente a lo largo del sistema.
  - Cada vez que el sistema utiliza la persistencia, se realiza del mismo modo.
  - Al definir la persistencia como un mecanismo arquitectónico, se ofrece un método por defecto que todos los diseñadores deben seguir e implementar de un modo consistente.

Mecanismos Arquitectónicos

- Las infraestructuras sobre las cuales se construirá el sistema y se deben definir en la arquitectura son:
  - *persistencia*
  - *distribución*
  - *comunicación*
  - *gestión de transacciones*
  - *seguridad*



Crear una arquitectura significa...

- Transformar los problemas de negocio en soluciones técnicas, lo que implica responsabilidades tanto tecnológicas como de gestión.

R. Tecnológicas	R. de Gestión

• identificar los casos de uso significantes para la arquitectura
• guiar el desarrollo de prototipos
	• ofrecer información y ayuda para la gestión de costes
	• gestionar las comunicaciones con el equipo de desarrollo para
	  • refinar y clarificar los requisitos
	  • generar confianza en los clientes/accionistas
	  • formar a los miembros del equipo

Arquitectura vs Diseño I

- ¿Cuando se acaba la creación de la arquitectura y cuando comienza el proceso de diseño?

	Arquitectura	Diseño
Nivel de Abstracción	Visión amplia y alta en pocos aspectos	Atención detallada y en profundidad de muchos aspectos
Entregables	Planos de sistema y subsistemas, y prototipos arquitectónicos	Diseño de componentes, especificaciones de código
Área de Atención	Requisitos no funcionales	Requisitos funcionales

Arquitectura vs Diseño II

- La arquitectura define qué es lo que se va a construir, y el diseño perfila cómo se va a hacer.
- La arquitectura se controla por una o pocas personas que se centran en la visión global y en cambio, el diseño por muchas personas que se centran en los detalles de cómo lograr la visión global.
- Un arquitecto crea una arquitectura hasta el punto que el equipo de diseño la puede utilizar para hacer que el sistema logre todos sus objetivos.
  - Por lo tanto, si hemos de crear una arquitectura para desarrolladores expertos, quizás no entraremos tan en detalle como si tuvieses a desarrolladores con menos experiencia.

Presupuesto

- Conforme se crea una arquitectura el *presupuesto* para adquirir hardware, software y recursos de desarrollo es limitado, de modo que el sistema debe desarrollarse bajo estas premisas.
  - ¿cómo podemos hacer que el sistema escale para lograr la demanda del cliente cuando solo tenemos un único ordenador para todos los servicios?
  - ¿cómo crear una arquitectura sin dinero para comprar productos software?

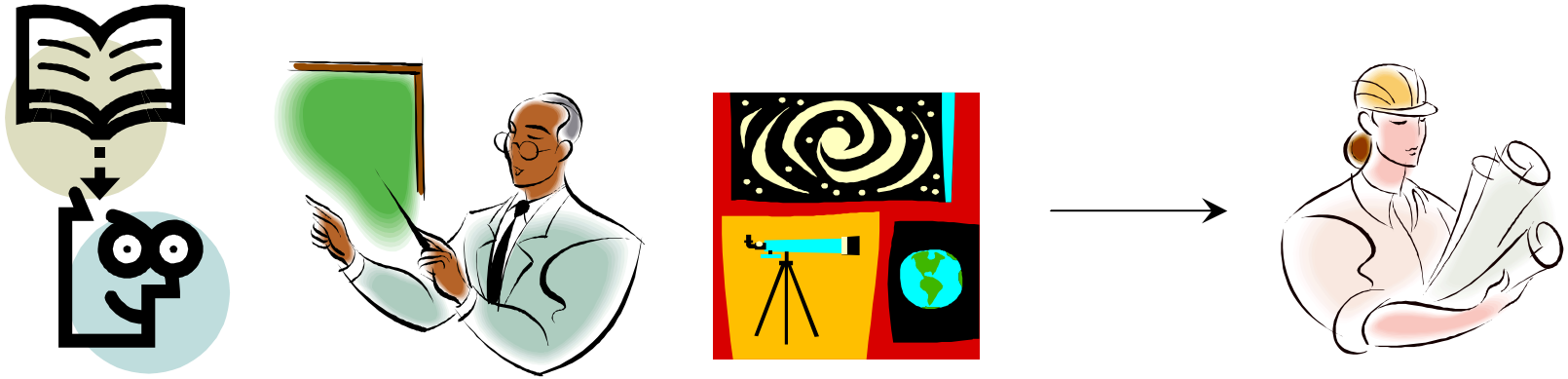
Equilibrio y Decisiones

- Conforme se superan las dificultades evaluando el correcto equilibrio, es importante que cada decisión tomada respecto a la arquitectura quede registrada.
  - Si tomamos una decisión sobre instalar una base de datos Oracle para persistir la información de la aplicación, debemos documentar esta decisión justificando porqué hemos elegido Oracle respecto a otros proveedores.
- Esto permite que el resto del equipo de desarrollo o integrantes que entran en fases tardías comprendan porqué se tomaron ciertas decisiones
  - previene el tener que justificar continuamente las decisiones tomadas.

Arquitecto...

- El arquitecto idóneo es aquella persona de letras, matemático, con estudios de historia, buen estudiante de filosofía, con gustos musicales, que no ignora la medicina, aprendiz de leyes, familiarizado con la astronomía y cálculos astronómicos.

Vitruvio, 25 AC

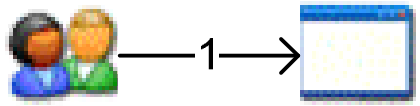


Arquitecto Software

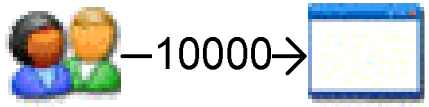
- Un arquitecto es una persona polifacético, madura, con experiencia, educada, que aprende de forma rápida, un líder con dotes de mando y buena comunicación, pudiendo tomar decisiones en los momentos difíciles.
  - Para que un arquitecto sea polifacético, debe tener conocimiento del negocio o del dominio del problema. Puede obtener este conocimiento mediante la experiencia o la formación
- Debe tener un dominio alto de las tecnologías.
  - Un arquitecto puede ser un experto es una tecnología particular, pero al menos debe tener conocimientos general de todas las tecnologías competentes respecto a qué tecnología es mejor para tomar decisiones sobre una base sólida.
  - Un buen arquitecto evalúa todas las posibles soluciones de un problema independientemente de la tecnología a utilizar.

Arquitecto vs Desarrollador Senior

- Las tareas de un diseñador están relacionadas con lo que ocurre cuando un usuario pulsa un botón de una aplicación.



- En cambio, un arquitecto se tiene que preocupar de lo que ocurre cuando ese mismo botón lo pulsan 10.000 usuarios.



- Así pues, un arquitecto reduce los riesgos técnicos (elementos desconocidos, sin probar, ...) asociados con un sistema.

Capacidad de Liderazgo y Comunicación

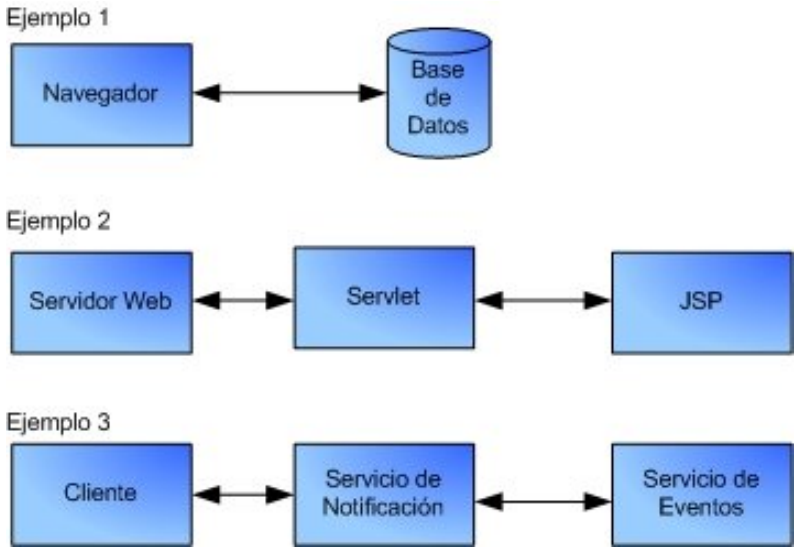
- El arquitecto debe liderar al equipo de desarrollo asegurándose que los desarrolladores y diseñadores construyen el sistema acorde a la arquitectura.
- Como líder, debe tomar decisiones sobre las dificultades del sistema. Para liderar, debe tener destreza en la escritura y ser un buen orador.
- Es responsabilidad del arquitecto transmitir el conocimiento del sistema (ya sea mediante modelos visuales o reuniones en grupo) a los desarrolladores, los cuales lo van a implementar.
- Si el arquitecto no se comunica de un modo efectivo, casi seguro que los desarrolladores no construirán el sistema correctamente.

Términos Arquitectónicos

- La arquitectura se refiere a la representación abstracta de los componentes y comportamientos de un sistema.
  - Una buena arquitectura de sistema reutiliza componentes, ya que cada componente se compone de partes que pueden estar repetidas, y por lo tanto, reutilizadas.
- La abstracción naturalmente forma capas que representan diferentes niveles de complejidad.
  - Cada capa describe una soluciones
- Estas capas se integran unas con otras de modo que las abstracciones de alto nivel se ven afectadas por las abstracciones de bajo nivel.

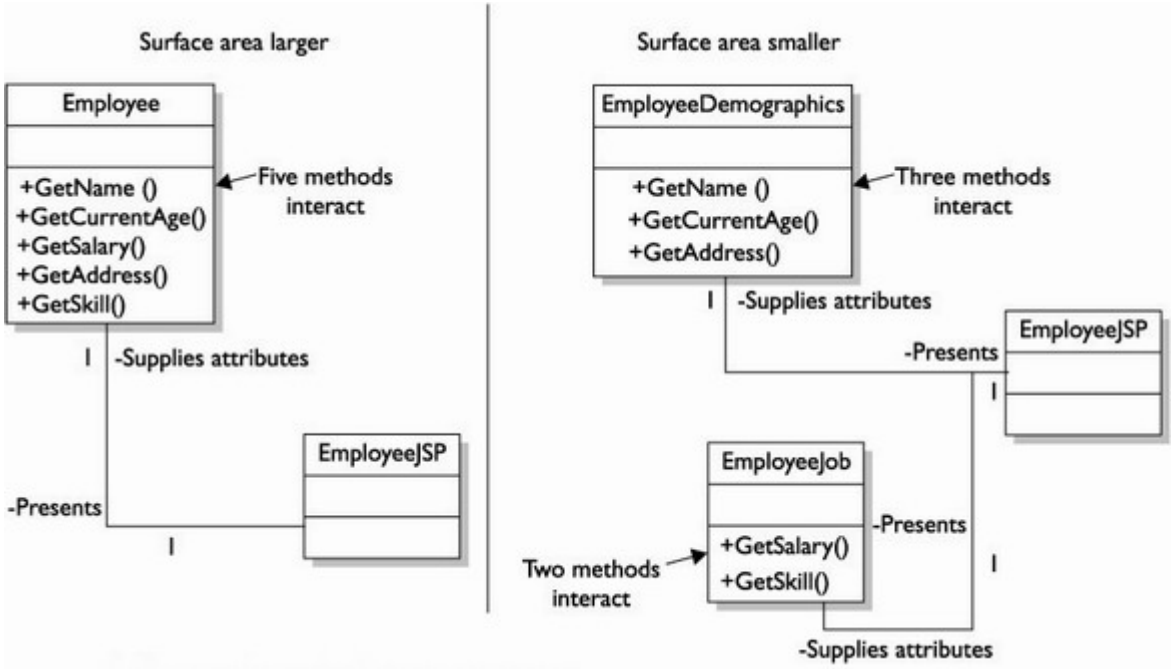
Abstracción

- El término **abstracción** implica el uso de una notación para algo titulizado de forma repetida en un diseño
  - oculta los detalles utilizando una representación clara
- Es el primer paso del proceso de diseño
  - realizando un proceso top-down para dividir el sistema de un modo jerárquico
  - examinando cada nivel de la jerarquía en términos de funciones e intenciones del diseño



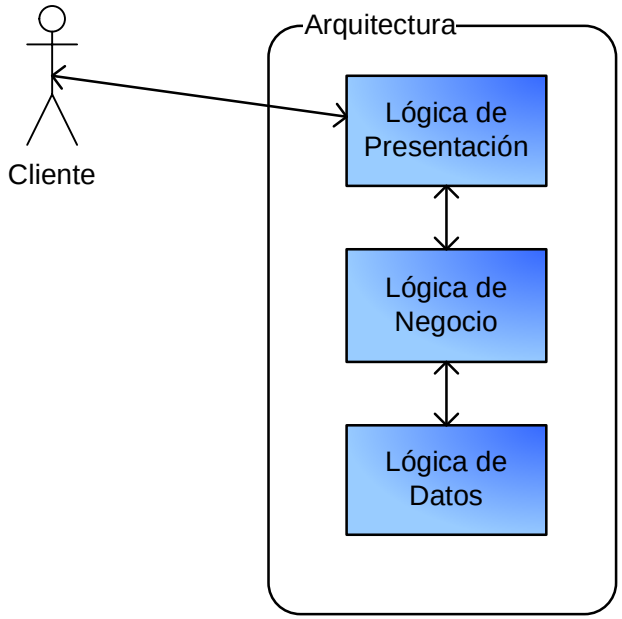
Área de Superficie

- Describe el modo en el cual los componentes interactúan unos con otros.
  - cuanto mayor sea esta área, más componentes se verán afectados por el cambio de un componente del área

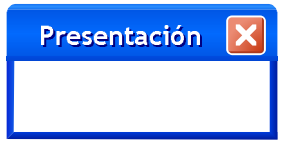


Arquitectura de 3 Capas

- Una arquitectura de 3 capas separa claramente las lógicas de **presentación**, **negocio** y **datos**.
- Cabe distinguir capa (capa lógica o *layer*) de nivel (capa física o *tier*).
- La separación de estas 3 capas es lógica, sin necesidad de que cada una de las capas se separe en un nivel.
  - Es decir, las 3 capas pueden residir sobre la misma máquina, pero a nivel lógico, estarán separadas.

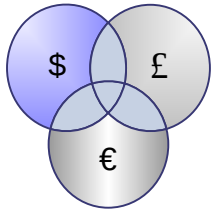


Lógica de Presentación



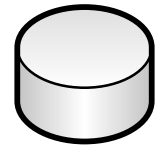
- La capa de presentación es responsable de presentar la información e interactuar con las capas inferiores
  - Muestra el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso
    - Realiza un filtrado previo para comprobar que no hay errores de formato
- Únicamente se comunica con la capa de negocio.
- ¿Quien trabaja en el interfaz de la aplicación?
  - Diseñadores y programadores de interfaces de usuario
  - Los programadores que programan la interfaz no suelen ser son los mismos que programan las capas de negocio y datos.
  - Además, utilizan diferentes aplicaciones para hacer su trabajo.
- Objetivo → la gente de presentación trabaje con sus archivos (docs html, hojas de estilo, imágenes, etc...) , mientras que la gente de negocio con los suyos (fichero java, ant, etc...).

Lógica de Negocio



- La capa de negocio es la responsable de implementar las operaciones solicitadas por los clientes a través de la capa de presentación.
  - Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.
- Se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para almacenar o recuperar datos del SGBD.

Lógica de Datos



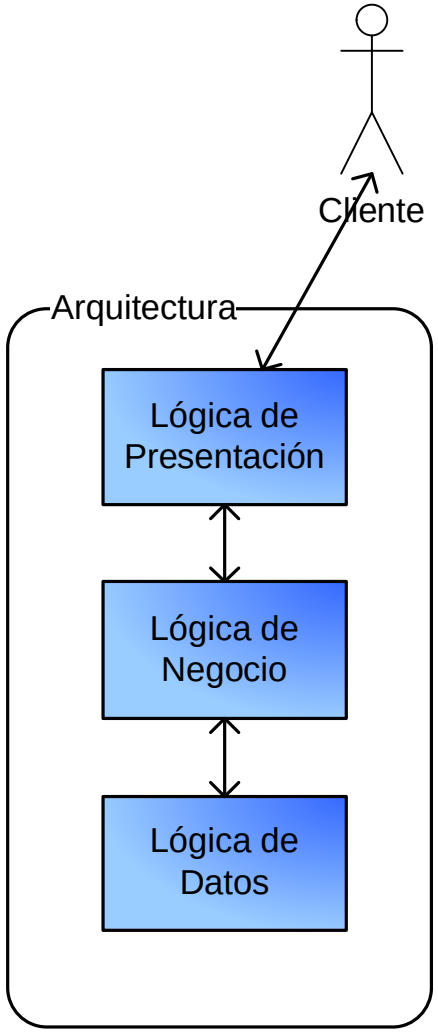
- La capa de **datos** es la responsable de gestionar todos los elementos de información del sistema
  - Ficheros planos, XML, SGBD, etc...
- Algunas arquitecturas consideran parte de esta capa aquellos sistemas externos que proporcionan información
  - Servidores de mensajes, servicios web, etc...
- El principal motivo para separar los datos, y en concreto la base de datos es
  - Como las pilas, se venden por separado.
  - Es un elemento crítico de una aplicación, ya que si falla la base de datos, la aplicación se cae.
  - La información debe estar lo más segura posible

    Candidata ideal a separarla de la aplicación y de los clientes, con un gran cortafuegos impidiendo el acceso a extraños.

Diseño Top-Down

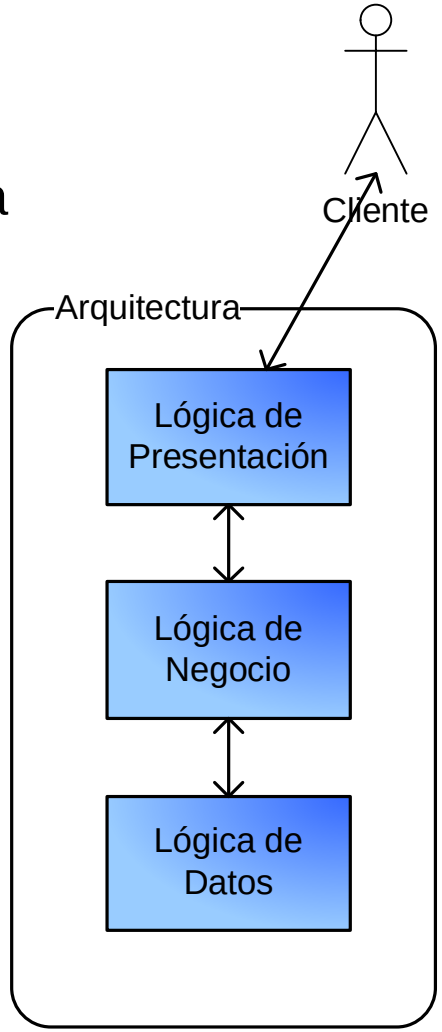
- Se define la funcionalidad del sistema desde el punto de vista del cliente
- Las funcionalidades se propagan por las capas según las necesidades identificadas en las capas anteriores

- Favor
  - Desde el principio se tienen claras las funcionalidades, y éstas dirigen el desarrollo del sistema
- En Contra
  - Sólo es posible aplicarlo a sistemas desarrollados desde cero.
  - Los componentes acaban fuertemente acoplados pues se utilizan en entornos homogéneos



Diseño Bottom-Up

- Surge más por necesidad que por elección
  - Muchos de los sistemas actuales se basan en la integración de productos existentes (*legacy* o heredados)
    - Un sistema heredado es aquel que es utilizado en un contexto distinto del que en principio fue concebido
  - Si tenemos que integrar sistemas heredados no podemos seguir un enfoque top-down.
- El cliente define las funcionalidades, y dependiendo de los recursos existentes y las funcionalidades que ofrecen, se encapsulará aquella funcionalidad existente, adaptando las salidas de la aplicación a las necesidades del cliente.

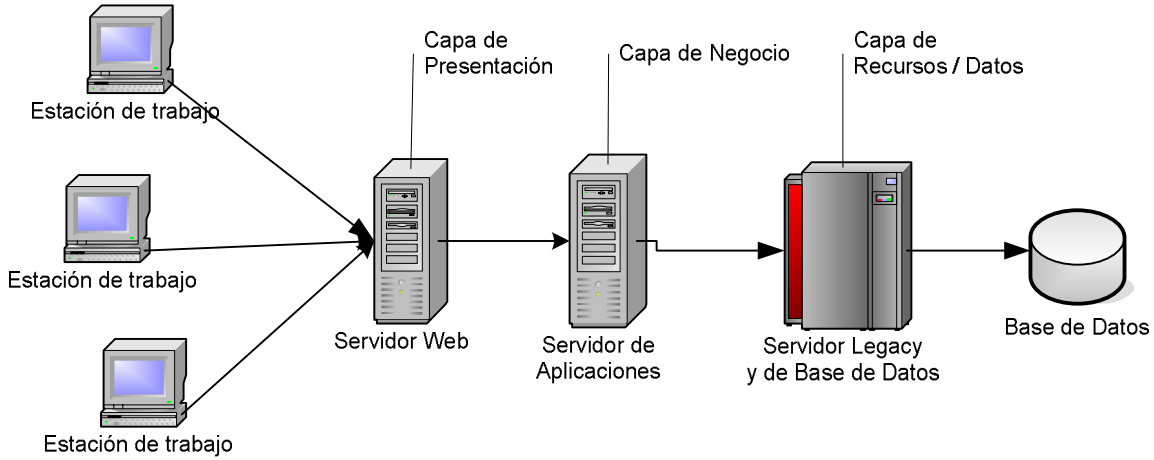


Capa / Layer

- Las capas son sistemas lógicos por si mismos, y hacen lo mismo que todos los sistemas: interactuar con el entorno para obtener entradas y producir salidas.
- Tipos de Sistemas con capas
  - bidireccionales → ofrecen servicios a sus capas superiores e inferiores
  - unidireccionales → ofrecen servicios a las capas superiores mientras que utilizan los servicios de las capas inferiores
- Mediante un uso estricto de las capas, las clases y objetos de una capa deben depender de clases u objetos dentro de la misma capa o inferior.
  - Construir una capa y sus objetos de esta manera hace posible construir las capas inferiores antes que las superiores.

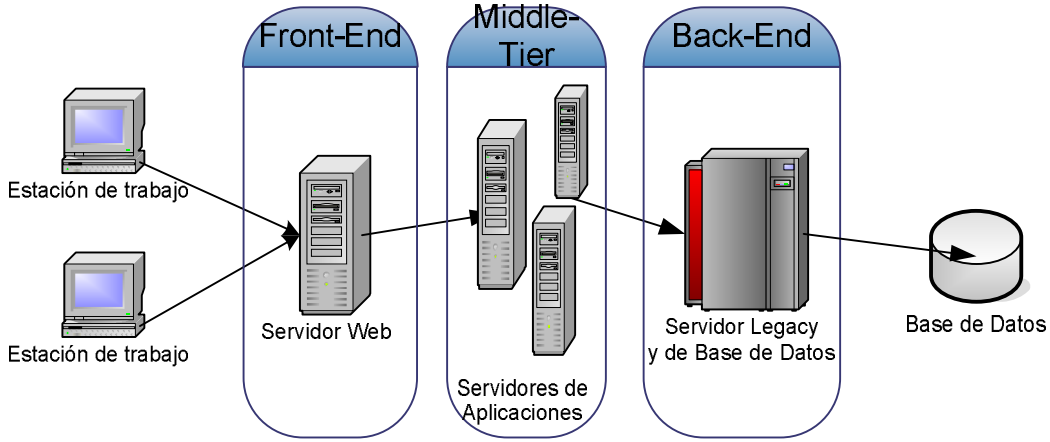
Nivel / Tier

- En un entorno de múltiples niveles (también conocidos como capas físicas), las capas lógicas se reparten en uno o más niveles.
  - el cliente implementa la lógica de presentación (cliente ligero)
  - la lógica de negocio se implementa sobre el servidor de aplicaciones (1 o varios)
  - los datos residen en el servidor de base de datos (1 o varios).



Arquitectura Multinivel

- El componente de **front-end**
  - ofrece portabilidad en la lógica de presentación (servidor web)
- El componente de **back-end**
  - ofrece acceso a servicios dedicados (servidor de base de datos)
- Uno o más componentes de **middle-tier**
  - permite a los usuarios compartir y controlar la lógica de negocio, aislándola de la aplicación real (servidor de aplicaciones)



Ventajas Arquitectura Multinivel I

- Los cambios en el interfaz de usuario o en la lógica de la aplicación son independientes unos de otros
  - permite a la aplicación evolucionar fácilmente para cumplir nuevos requisitos
- Se minimizan los cuellos de botellas debido a problemas de red, ya que la capa de la aplicación no transmite información extra al cliente
  - únicamente transmite aquella información que realmente es necesaria para realizar una tarea
- Cuando son necesarios cambios en la lógica de negocio, solo debemos actualizar el servidor.
  - En una arquitectura de 2 niveles, debemos modificar cada cliente cuando tenemos un cambio en negocio.

Ventajas Arquitectura Multinivel II

- Se aísla al cliente de la BBDD y las operaciones de red
  - El cliente puede acceder a los datos fácilmente sin necesidad de saber donde están los datos o cuantos servidores existen.
- Se pueden reutilizar las conexiones de BBDD, mediante un *pool de conexiones*
  - las conexiones se comparten entre varios usuarios
  - reduce drásticamente los costes asociados a las licencias por usuarios.
- Se consigue independencia de la organización respecto de los datos
  - la capa de datos utiliza SQL estándar
  - la empresa no se ata a procedimientos almacenados específicos de un determinado SGBD.

Requisitos No Funcionales

- También conocidos como requisitos a nivel de servicio o requisitos de servicios de calidad (*QoS*).
- La arquitectura debe tratar los siguientes requisitos no funcionales:
  - rendimiento
  - escalabilidad
  - confiabilidad
  - disponibilidad
  - extensibilidad
  - mantenibilidad
  - gestionabilidad
  - seguridad
- Como arquitecto, hay que buscar el equilibrio entre estos requisitos.
  - Por ejemplo, si el requisito más importante es el rendimiento de un sistema, quizás haya que sacrificar la mantenibilidad y extensibilidad del sistema para asegurar los mínimos de calidad esperados respecto al rendimiento.

Rendimiento

- Tiempos de respuesta por transacción de usuario.
- Otra medida → cuantos usuarios simultáneos soporta un sistema dentro de unos tiempos de respuesta razonables
  - No se empleen más de tres segundos en cada formulario de pantalla
- **Productividad** → número de transacciones que se realizan en un periodo de tiempo determinado, normalmente un segundo.
  - El sistema debe soportar 100 transacciones en un segundo.
- Necesitamos crear una arquitectura que permita a los diseñadores y desarrolladores a completar el sistema sin necesidad de considerar estas métricas de rendimiento.
- Para conseguir un buen rendimiento, se necesita un buen diseño y una buena codificación
  - Otros factores claves son la memoria y la velocidad del procesador de la CPU.

Escalabilidad

- Habilidad de soportar los servicios de calidad conforme la carga del sistema crece sin que el sistema se vea perjudicado y sin necesidad de modificar el mismo.
  - Un sistema se considera escalable si, conforme se incrementa la carga, el sistema sigue respondiendo dentro de unos limites aceptables.
    - Si no responde dentro de los limites, que sea escalable no significa que tengamos que modificar el código.
    - Si es escalable, con sólo instalar más hardware o instancias de los servidores, el sistema debe recuperar los tiempos de respuesta esperados.

- **Capacidad** del sistema → número máximo de procesos o usuarios que un sistema puede manejar manteniendo la calidad de los servicios ofrecidos.
  - Un sistema se ejecuta a su capacidad máxima cuando ya no puede responder a más peticiones dentro de unos tiempos de respuesta aceptables.

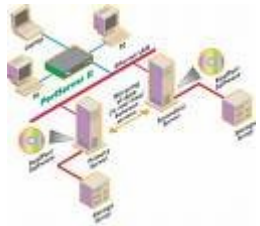
Escalado vertical y horizontal

- Para escalar un sistema que ha llegado a su capacidad, se debe añadir hardware adicional.
- El **escalado vertical** implica añadir procesadores, memoria, o discos adicionales a la/s máquina/s actual/es.
- El **escalado horizontal** implica añadir más máquinas al entorno (cluster), de este modo se incrementa la capacidad general del sistema.
- La arquitectura debe manejar el escalado vertical y horizontal del hardware
- El escalado vertical es más fácil que el horizontal.
  - Añadir más procesadores o memoria normalmente no tiene un impacto en la arquitectura
  - Cuando la arquitectura debe ejecutarse sobre múltiples máquina pero que parezca que sólo se ejecuta como un sistema compacto es más difícil.

Confiabilidad

- Asegura la integridad y consistencia de la aplicación y de todas sus transacciones.
  - Conforme crece la carga del sistema, éste debe continuar procesando las peticiones y manejando las transacciones con la misma precisión que lo hacia antes de incrementarse la carga.
- Puede tener un impacto negativo en la escalabilidad
  - si el sistema no puede mantener la confiabilidad conforme la carga se incrementa, entonces el sistema no es escalable.
- **Tolerancia a fallos** → capacidad de un sistema para seguir funcionando cuando se produce un error.
  - Para conseguir tolerancia a fallos hay que diseñar el sistema con un alto grado de redundancia de hardware.
  - Si se produce un error en un único componente, el componente redundante asumirá su función sin que se produzca un tiempo de inactividad apreciable

Disponibilidad



- Asegura que un servicio/recurso esta siempre accesible
  - o parece que lo está
- La confiabilidad puede contribuir a la disponibilidad, pero la disponibilidad se puede alcanzar incluso si algunos componentes fallan.
- Montando un entorno de componentes redundantes en cluster y tolerantes a fallos, un componente puede caerse y tener un impacto negativo en la confiabilidad, pero el servicio sigue en pie gracias a la redundancia.
  - En cuanto un componente se cae, el componente redundante suplanta al componente caído.

Extensibilidad

- Habilidad para añadir funcionalidad adicional o modificar funcionalidad existente sin tener un impacto directo en la funcionalidad del sistema.
  - La extensibilidad no se puede medir cuando el sistema está desplegado, sino cuando se debe extender por primera vez.
- Al crear la arquitectura debemos tener en cuenta:
  - el bajo acoplamiento
  - el uso de interfaces,
  - la encapsulación
  - el diseño basado en patrones.
- Hemos de vigilar muy de cerca el código fuente de la aplicación → debe ser homogéneo y de calidad.
  - El equipo de desarrollo debe ser un equipo y no un conjunto de individualidades, donde la calidad del código fuente sea un objetivo común.

Mantenibilidad

- Habilidad de corregir defectos en la funcionalidad existente sin tener impacto en otros componentes del sistema.
  - Tampoco se puede medir a la hora del despliegue.
- Igual que en la extensibilidad, cuando creamos la arquitectura, para mejorar la mantenibilidad debemos tener en cuenta conceptos como:
  - bajo acoplamiento
  - alta cohesión
  - modularidad
  - documentación.
- La escalabilidad es uno de los enemigos de la mantenibilidad, ya que la complejidad asociada con los componentes distribuidos reduce la mantenibilidad.

Gestionabilidad

- Habilidad del sistema de asegurar la salud continua del sistema respecto a la escalabilidad, confiabilidad, disponibilidad, rendimiento y seguridad.
- La gestionabilidad se ocupa de la **monitorización** del sistema para mejorar los QoS dinámicamente sin modificar el sistema.
- La arquitectura debe tener la habilidad de monitorizar el sistema y permitir los cambios de configuración del sistema en caliente.
  - Llos servidores de aplicaciones ofrecen herramientas de monitorización

Seguridad

- Habilidad de asegurar que el sistema no se vulnera
  - que todos hacen lo que deberían hacer y nadie hace lo que no debería
- Es la cualidad más difícil de conseguir
- No solo trata temas como la confidencialidad o la integridad, sino también los ataques de denegación de servicio (DoS).
- Crear una arquitectura cuyos componentes están separados según funcionalidades es más fácil de proteger
  - se pueden construir zonas protegidas alrededor de dichos componentes mediante el uso de ACLs o cortafuegos
- Si se ataca a un componente, entonces es más fácil contener la amenaza de violación a ese único componente, evitando que se propague el ataque.

Principio de Oportunidad

- El principio de oportunidad normalmente rige los plazos de entrega del aplicativo, pero no por ello debe condicionar la calidad del proyecto.
  - La solución a entregar debe estar disponible dentro de un margen de tiempo que siga manteniendo el principio de oportunidad
  - El exceso de ingeniería puede penalizar la oportunidad del proyecto, entregando el proyecto tan tarde que ya no tenga ningún valor para el negocio del cliente.
- Aunque el arquitecto debe considerar todo los requisitos del cliente, también tiene la responsabilidad de avisar al cliente de las posibles implicaciones en el sistema que pueda tener el comprometer la arquitectura para llegar a una fecha de entrega muy cercana.

Arquitectura Ágil

- Una arquitectura ágil es aquella que conserva el principio de oportunidad, apropiada para los intereses del cliente y acelera la entrega del sistema.
- Es una arquitectura que se implementa fácil y eficientemente por el equipo de desarrollo, y de hecho, completada en un periodo de tiempo aceptable para el cliente.

Virtudes del Equipo

- Los miembros de un equipo de desarrollo tienen habilidades y conocimientos diferentes.
- La arquitectura debe sacar partido de estas diferencias.
- Los detractores piensan que la arquitectura debe basarse en la tecnología que mejor afronte los requisitos.
  - Si las tecnologías se hubiesen basado siempre en los conocimientos del equipo, la mayoría de las aplicaciones continuarían implementándose en COBOL.
  - Sin embargo, la experiencia técnica colectiva se debe tener en cuenta cuando el tiempo es un factor crítico en el proyecto.
- La misma regla debe aplicarse al buscar perfiles de desarrollo con unos ciertos conocimientos.
  - Si los miembros del equipo abandonan, o se debe formar un nuevo equipo, el completar dichas vacantes puede llegar a ser muy problemático si se requieren conocimientos en una tecnología desconocida.

Utilizar los Mejores Frameworks

- JavaEE esta lleno de frameworks, tanto como la plataforma J2EE es de extensa.
- Un framework de software apropiado puede reducir de forma significativa la cantidad de código que necesitas escribir y mejorar la calidad del diseño.
- Existen multitud de frameworks de código abierto para cada aspecto de una aplicación, incluyendo:
  - **Desarrollo web**: *Struts, Spring, JSF, Tapestry*
  - **Persistencia**: *Hibernate, iBatis, OJB*
  - **Entornos de construcción**: *Ant, CruiseControl, Maven*
  - **Entornos de pruebas**: *JUnit, Cactus*

Elegir un Framework I

- La elección de un framework es una decisión muy importante
  - Condiciona la arquitectura y el diseño de la aplicación
- Conocimiento del equipo
  - *Frameworks* probados con éxito en otros proyectos
  - Aquellos en los que el equipo de desarrollo ya tenga experiencia.
- Madurez
  - ¿Cuanto tiempo lleva utilizándose en el desarrollo de software comercial?
  - ¿Es un proyecto de código abierto mantenido por Apache, o el resultado de la tesis de un doctorado?
  - Un producto maduro ya está limado, y casi sin incidencias.

Elegir un Framework II

- Apto para el propósito
  - Apropiado para las necesidades específicas del sistema
  - Cuando no se encuentra una elección acertada, el desarrollo de una solución propia para un área de negocio determinada puede obtener beneficios a largo plazo.
- Herramientas de soporte
  - La combinación del framework junto a las herramientas de desarrollo pueden ofrecer ganancias de productividad significativas.
- Longevidad
  - Seleccionar aquellos framework que tendrán soporte durante el tiempo de vida prevista para el proyecto.
  - Un framework que desaparece del horizonte una vez el sistema alcanza la producción presenta graves problemas de soporte y mantenimiento.

Ser Previsor

- Ser previsor durante el proceso de diseño facilita en gran medida la vida de los desarrollador y probadores.
- Especialmente relevante en el área de pruebas
  - Cualquier acción que tome el arquitecto para facilitar las pruebas del sistema
    - reportará calidad al sistema
    - reducirá los esfuerzos totales en este aspecto.
- Debemos evitar ser previsores en exceso
  - un diseño pensado para una reutilización es mucho más complejo que un diseño simple.
  - Cuando el tiempo premia, se debe evitar la sobreingeniería y limitarse a lo necesario.

Si Queremos Ser Previsores Podemos...

- *Incluir esqueletos de pruebas y pruebas unitarias como parte del diseño*
  - En el desarrollo dirigido por las pruebas, se utiliza mucho tiempo codificando las pruebas.
  - Si el arquitecto es previsor y diseña/implementa los esqueletos de las pruebas, los desarrolladores sólo deben complementarlas con la casuística de éstas.
- *Evitar el uso de interfaces débilmente definidos*
  - cuando un método recibe un parámetro como ...
    - un String, el cual hay que parsear para obtener la información
    - un objeto muy grande, donde no necesitamos la mitad de los atributos

  ... provoca que el método sea excesivamente ambiguo, sin saber que datos tiene de entrada y cuales no

Diseño Ortogonal

- Ortogonalidad es sinónimo de software bien construido, con bajo acoplamiento y alta cohesión
  - Los componentes ortogonales son independientes, autocontenidos, y tienen una responsabilidad claramente definida.
  - Los cambios en el interior de un componente no tienen repercusiones en el resto de componentes del sistema
  - Los componentes ortogonales son fáciles de codificar, probar, y mantener gracias a su diseño
- La independencia entre los componentes es vital para una arquitectura ágil, ya que los cambios en un componente/capa de la aplicación no tiene por que propagarse por el sistema.
- Uno de los elementos principales dentro de la OO para alcanzar un diseño ortogonal es el uso de interfaces.
  - Si diseñamos nuestros componentes mediante interfaces, podemos utilizarlos como escudos contra los cambios de implementación de los métodos.

Roadmap → Puntos Destacados I

- La *arquitectura* es una representación abstracta del comportamiento y componentes del sistema.
  - Una buena arquitectura tiende a reutilizar componentes porque cada componente se divide en partes que pueden repetirse, y por tanto, reutilizar.
  - La abstracción forma capas que representan diferentes niveles de complejidad.
- La principal diferencia entre los términos *arquitectura y diseño* es el nivel de detalle.
  - La arquitectura opera a un nivel de abstracción alto con pocos detalles.
  - El diseño opera a bajo nivel de abstracción, pero con más atención en los detalles de la implementación.
- Las *capas* de una arquitectura son sistemas por si mismos.
  - Obtienen entradas de su entorno y ofrecen salidas al mismo.

Roadmap → Puntos Destacados II

- Los requisitos no funcionales o capacidades de una arquitectura incluye:
  - *Rendimiento*: habilidad de ofrecer la funcionalidad dentro de unos periodos de tiempo asumibles por los objetivos especificados.
  - *Disponibilidad*: grado de accesibilidad de un sistema. El término 24×7 describe la disponibilidad total. Este aspecto del sistema esta frecuentemente acoplado con el rendimiento.
  - *Escalabilidad*: habilidad de soportar el rendimiento y la disponibilidad requerida conforme crece la carga de transacciones.
  - *Confiabilidad*: habilidad de asegurar la integridad y consistencia de la aplicación y sus transacciones.
  - *Extensibilidad*: habilidad de extender la funcionalidad del sistema.

Roadmap → Puntos Destacados III

- *Gestionabilidad*: habilidad de administrar y gestionar los recursos del sistema para asegurar la disponibilidad y rendimiento respecto a otra capacidades.
- *Flexibilidad*: habilidad de manejar cambios en las configuraciones hardware y arquitectónicas sin un gran impacto en los sistema dependientes.
- *Capacidad*: habilidad del sistema de ejecutar múltiples tareas por unidad de tiempo.
- *Validez*: habilidad de predecir y confirmar los resultados basados en una entrada especificado o un gesto de un usuario.
- *Reusabilidad*: habilidad de utilizar un componente en más de un contexto sin necesidad de cambios internos.
- *Seguridad*: habilidad de asegurar que no se accede ni modifica la información a no ser que se cumplan las políticas de empresa.

Roadmap → Certificación Sun

- SCEA cita 2 objetivos (sección 2 - Arquitecturas Comunes):
  - Reconocer el efecto de cada una de las características de una arquitectura de 2, 3 o N niveles: escalabilidad, mantenibilidad, confiabilidad, disponibilidad, extensibilidad, rendimiento, manejabilidad y seguridad
  - Dada una arquitectura definida en términos de organización de red, listar los beneficios y posibles debilidades asociadas.

- En la siguiente sesión veremos:
  - la arquitectura Java EE y los requerimientos de cada tecnología
  - el uso de los patrones de diseño dentro del marco empresarial, ubicando cada patrón en su correspondiente capa/nivel.

Roadmap → Para Saber Más

- **Bibliografía**
  - ***Software Architecture in Practice***, 2nd Edition, Addison-Wesley, de *Len Bass y otros*
  - ***Ingeniería del Software***, 7ª Edición, Addison-Wesley, de *Ian Sommerville*
  - ***Ingeniería del Software***, 6ª Edición, McGraw-Hill, de *Roger S. Pressman*
  - ***Sun Certified Enterprise Architect for J2EE Study Guide*** *(Exam 310-051)*, McGraw-Hill, de *Paul Allen y Joseph Bambara*
  - ***Sun Certified Enterprise Architect for J2EE Technology Study Guide***, Prentice-Hall, de *Mark Cade y Simon Roberts*
- **Enlaces**
  - Certificación Arquitecto Sun : http://www.sun.com/training/certification/java/scea.xml
  - Articulos sobre AS en Español realizados por MSDN http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.asp

¿Preguntas...?