

Arquitectura de Software

Definiciones y Contexto



Juan Bernardo Quintero

Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

Definiciones de Arquitectura de Software

Clements:

La Arquitectura de Software es a grandes rasgos, una vista del sistema que incluye los *componentes* principales del mismo, la conducta de esos *componentes* según se la percibe desde el resto del sistema y las formas en que los *componentes* interactúan y se coordinan para alcanzar la misión del sistema.

IEEE 1471-2000:

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus *componentes*, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

IEEE 610.12.1990:

Ingeniería de Software es La aplicación de una estrategia sistemática, disciplinada y cuantificable al desarrollo, aplicación y mantenimiento del software; esto es, la aplicación de la ingeniería al software.

<http://www.sei.cmu.edu/architecture/definitions.html>

La Arquitectura no es...

- Una normativa madura
- Igual en la academia y en la industria
- Diseño de software con UML
- Naturalmente vinculada con ingeniería & ciclo de vida
 - Ocurre en algún punto entre la elicitation de requerimientos y la especificación de casos de uso, o entre éstos y el diseño
- Naturalmente vinculada a metodología (RUP)
- Naturalmente relacionada con modelado Orientado a Objetos
- Hay vínculo “natural” entre requerimientos (casos de uso) y clases
- Las herramientas arquitectónicas generan el código de la aplicación

Arquitectura es...

- Vista estructural de alto nivel
- Define estilo o combinación de estilos para una solución
- Se concentra en requerimientos no funcionales
 - Los requerimientos funcionales se satisfacen mediante modelado y diseño de aplicación
- Esencial para éxito o fracaso de un proyecto

Corrientes principales

- Arquitectura estructural – SEI – Carnegie Mellon
 - Garlan, Shaw, Clements
 - Variantes con modelos de datos (Medvidovic), radicales, formales (Moriconi-SRI), etc
- Arquitectura como etapa de la ingeniería de software orientada a objetos
 - James Rumbaugh, Grady Booch, Ivar Jacobson (“los 3...”), Craig Larman...
- Arquitectura basada en patrones – SEI
 - Redefinición de estilos como patrones POSA
 - Microsoft Patterns & Practices
- Arquitectura procesual y metodologías
 - Kazman, Bass (SEI)
 - Variantes de arquitectura basada en escenarios

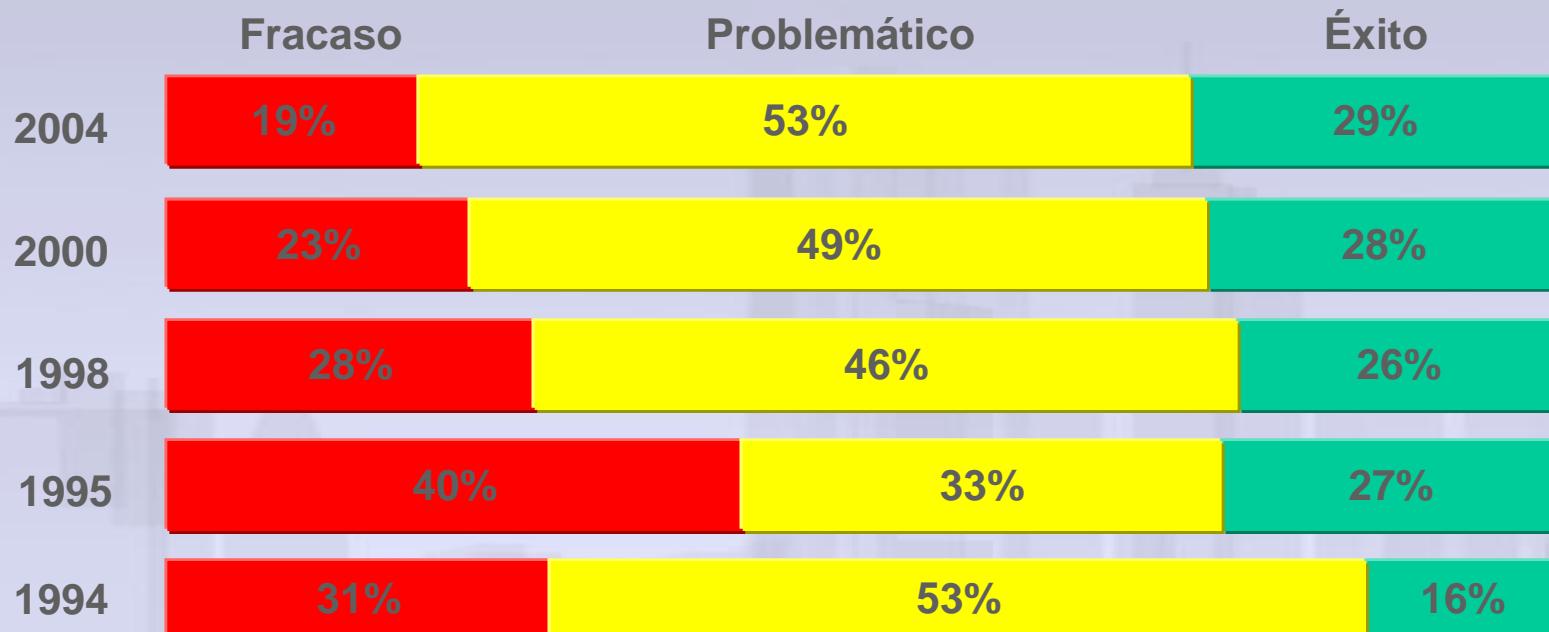
Arquitectura de Software

- Definir los módulos principales
- Definir las responsabilidades que tendrá cada uno de estos módulos
- Definir la interacción que existirá entre dichos módulos:
 - Control y flujo de datos
 - Secuenciación de la información
 - Protocolos de interacción y comunicación
 - Ubicación en el hardware

Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

Evolución del Desarrollo de Software Según el Reporte Caos



El proyecto se aborta o el sistema no se llega a utilizar



Desbordamiento de agendas o costes. Las funcionalidades no cubren las expectativas. Problemas funcionales



Proyecto realizado en el tiempo previsto, con los costes previstos, con la funcionalidad esperada y ofreciendo un funcionamiento correcto.

Éxito en Proyectos de Software Según el Standish Group



Top Ten Reasons for Success

- 1. User Involvement
- 2. Executive Management Support
- 3. Clear Business Objectives
- 4. Optimizing Scope
- 5. Agile Process
- 6. Project Manager Expertise
- 7. Financial Management
- 8. Skilled Resources
- 9. Formal Methodology
- 10. Standard Tools and Infrastructure

Origen de los Problemas

■ Increasing Volume

- Data
- Code
- Aspects (functional and non-functional)

■ Increasing evolutivity

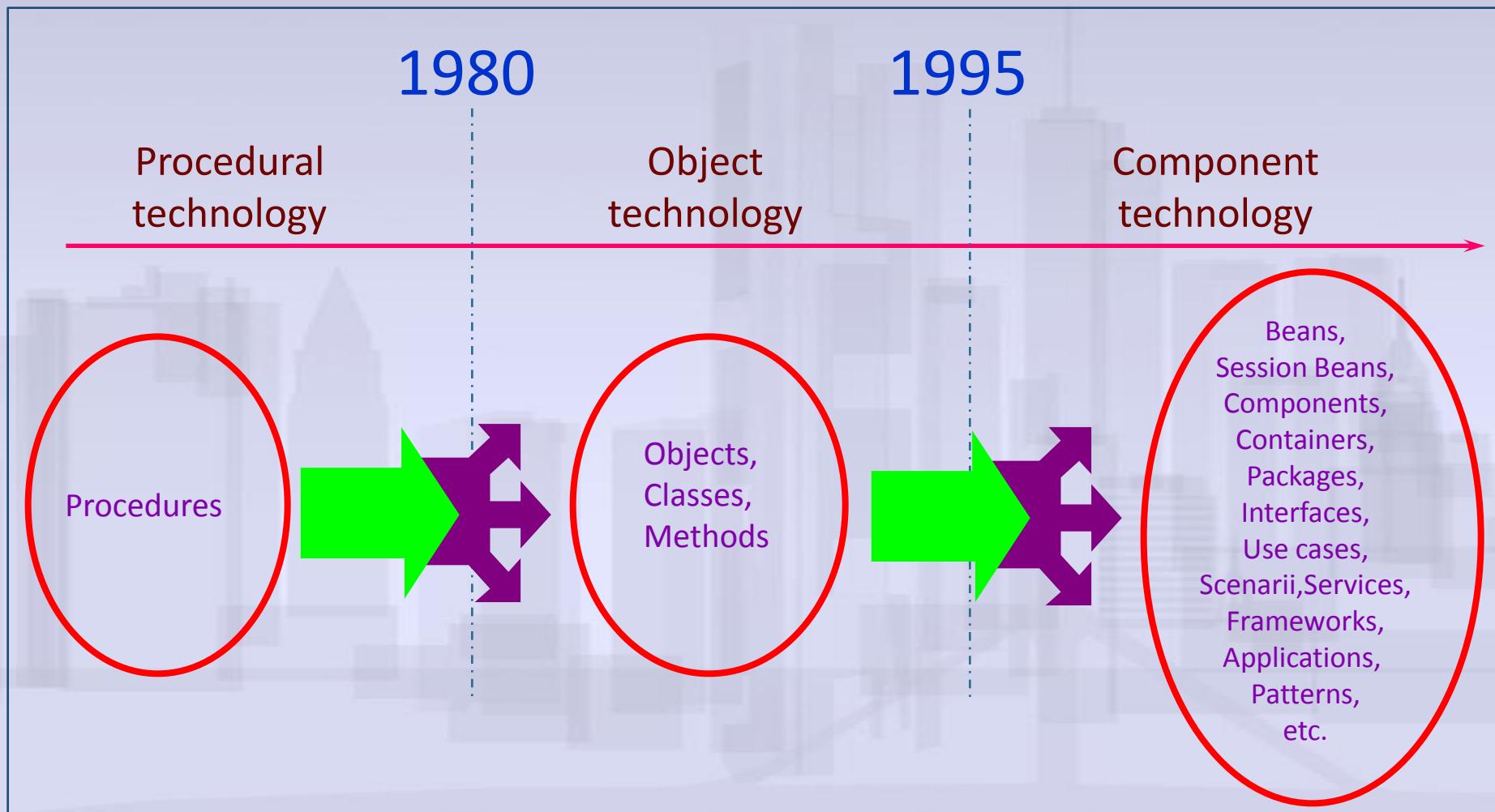
- of the business part (globalization, concentrations, reorganizations, increasing competition, ...)
- of the execution platform

■ Increasing heterogeneity

- languages and paradigms
- data handling and access protocols
- operating systems and middleware platforms
- technologies
 - The arrival rate of new technologies is increasing
 - This rate is not likely to decrease in the future
 - Old technologies don't die, they just hide within deep software layers

Origen de los Problemas

Simplifying or increasing complexity

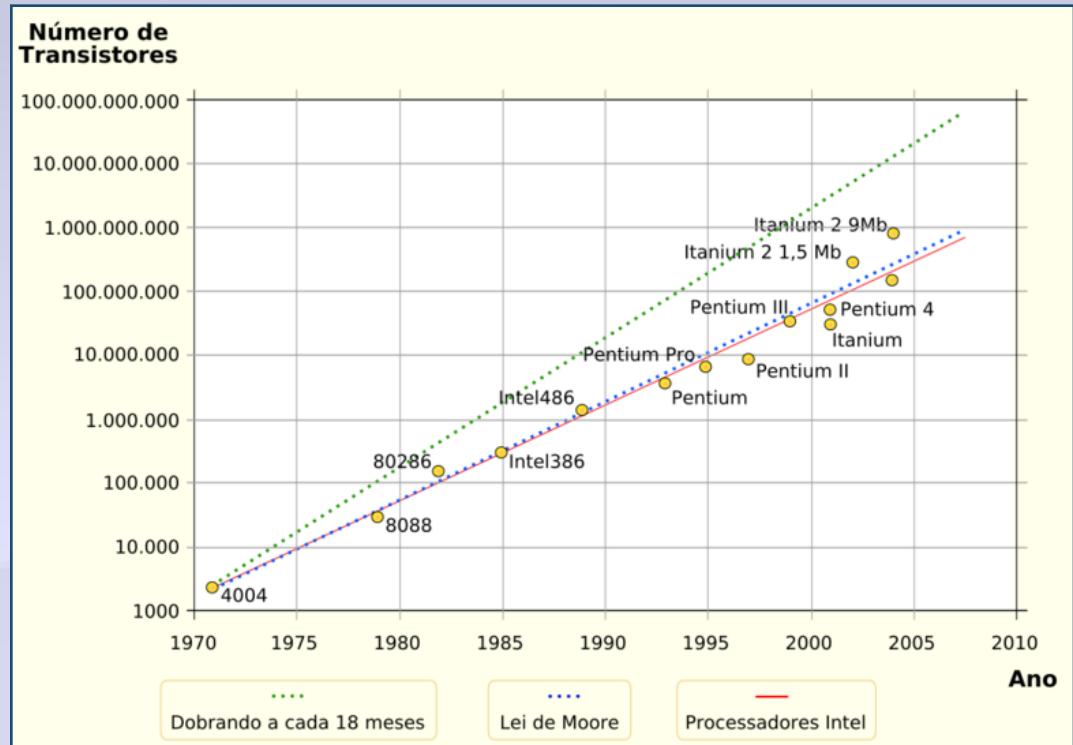


Evolución Acelerada del Hardware

- En la actualidad son cuatro los factores que imprimen un ritmo acelerado a la industria del hardware:
 1. Incremento constante de la capacidad de operación
 2. Miniaturización
 3. Reducción de costes para la producción de hardware
 4. Avance de las comunicaciones entre sistemas.
- La consecuencia es obvia:
 1. Ordenadores potentes
 2. Que pueden llevarse en el bolsillo
 3. En permanente conexión con grandes sistemas, redes de comunicación públicas, sistemas de localización GPS, etc.

Evolución Acelerada del Hardware

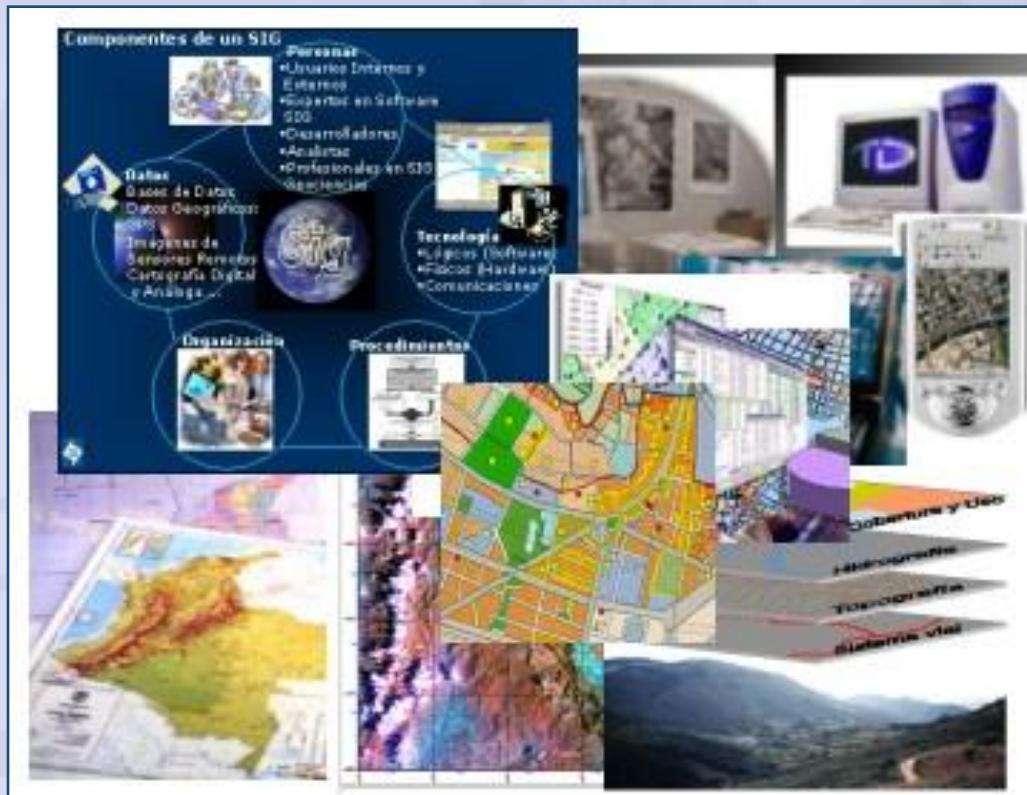
- La Ley de Moore expresa que aproximadamente cada dos años se duplica el número de transistores en un circuito integrado.
- Dicha ley explica la evolución del hardware
- Pero ¿Cuál es el efecto de la ley de Moore en el desarrollo de software?



En 1978, un vuelo comercial entre Nueva York y París costaba cerca de 900 dólares y tardaba 7 horas. Si se hubieran aplicado los mismos principios de la Ley de Moore a la industria de la aviación comercial de la misma forma que se han aplicado a la industria de los semiconductores desde 1978, ese vuelo habría costado cerca de un centavo de dólar y habría tardado menos de 1 segundo en realizarse.

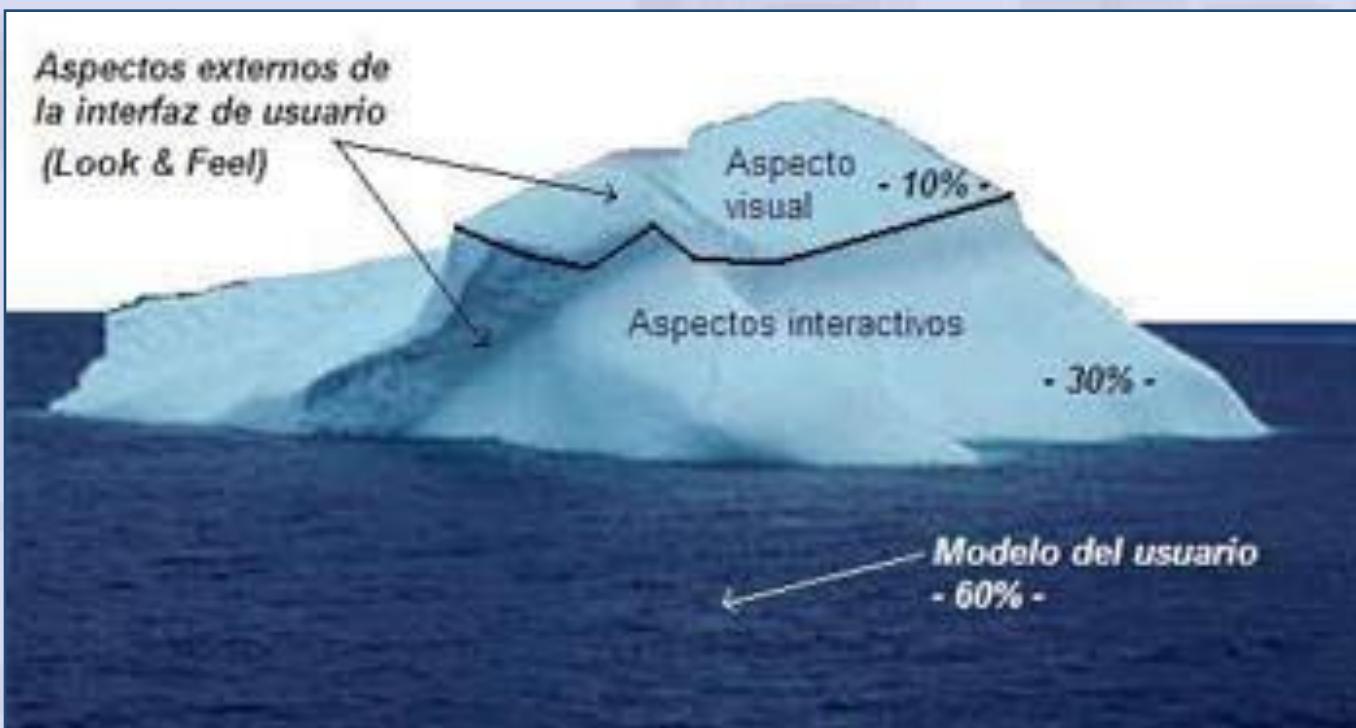
Los sistemas cada vez son más complejos (Efectos)

- Incremento de tasa de surgimiento de tecnologías
- Esta rata no parece decrementarse en el futuro
- “Las tecnologías viejas no mueren, solo se ocultan en capas mas profundas del software”



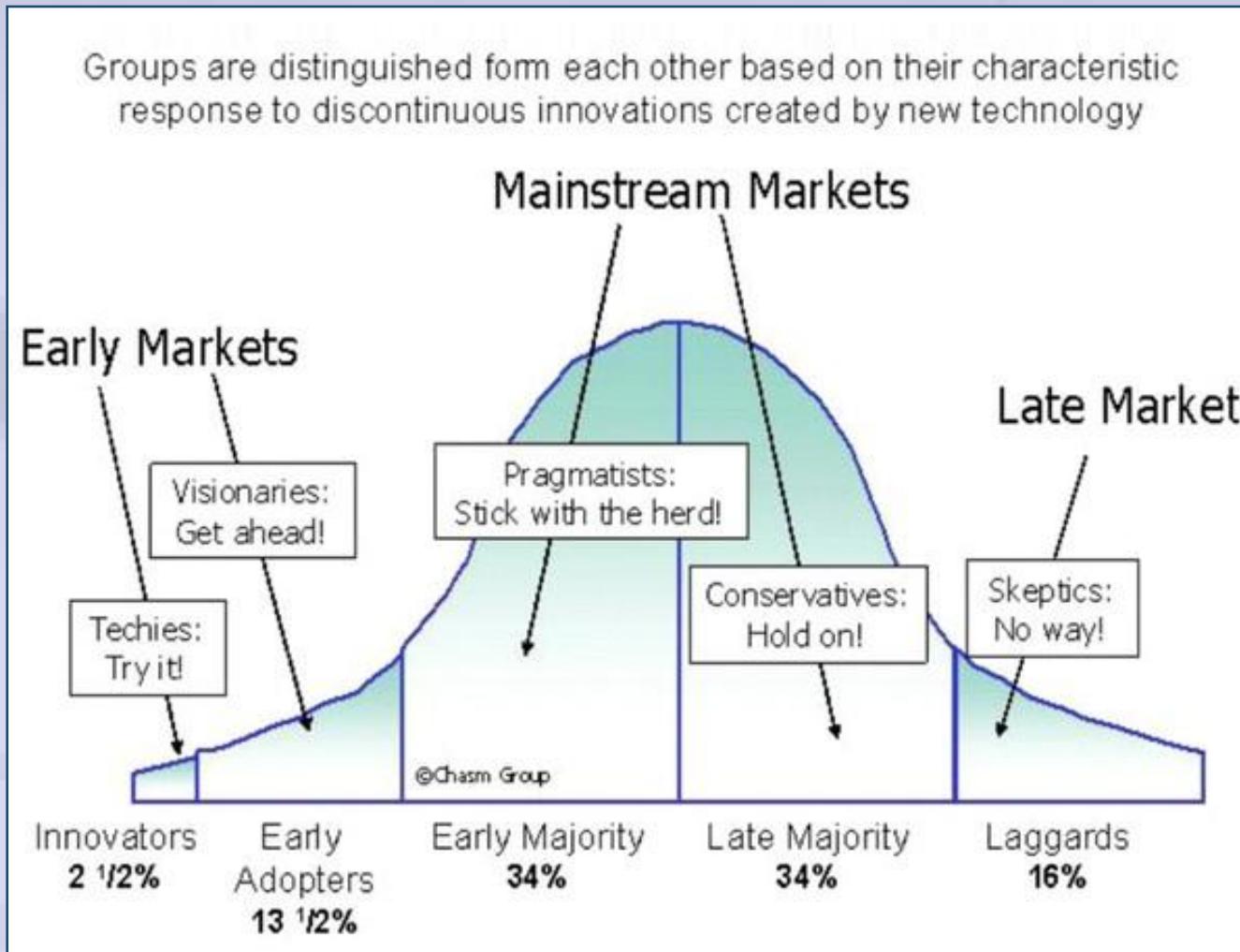
La Arquitectura de Software como disciplina (El Iceberg de la usabilidad)

- Según Dick Berry de IBM, el diseño de un sistema tiene 3 elementos:
 - la presentación de la información
 - la funcionalidad de la aplicación
 - la Arquitectura del Software



Adopción de la Arquitectura de Software como disciplina

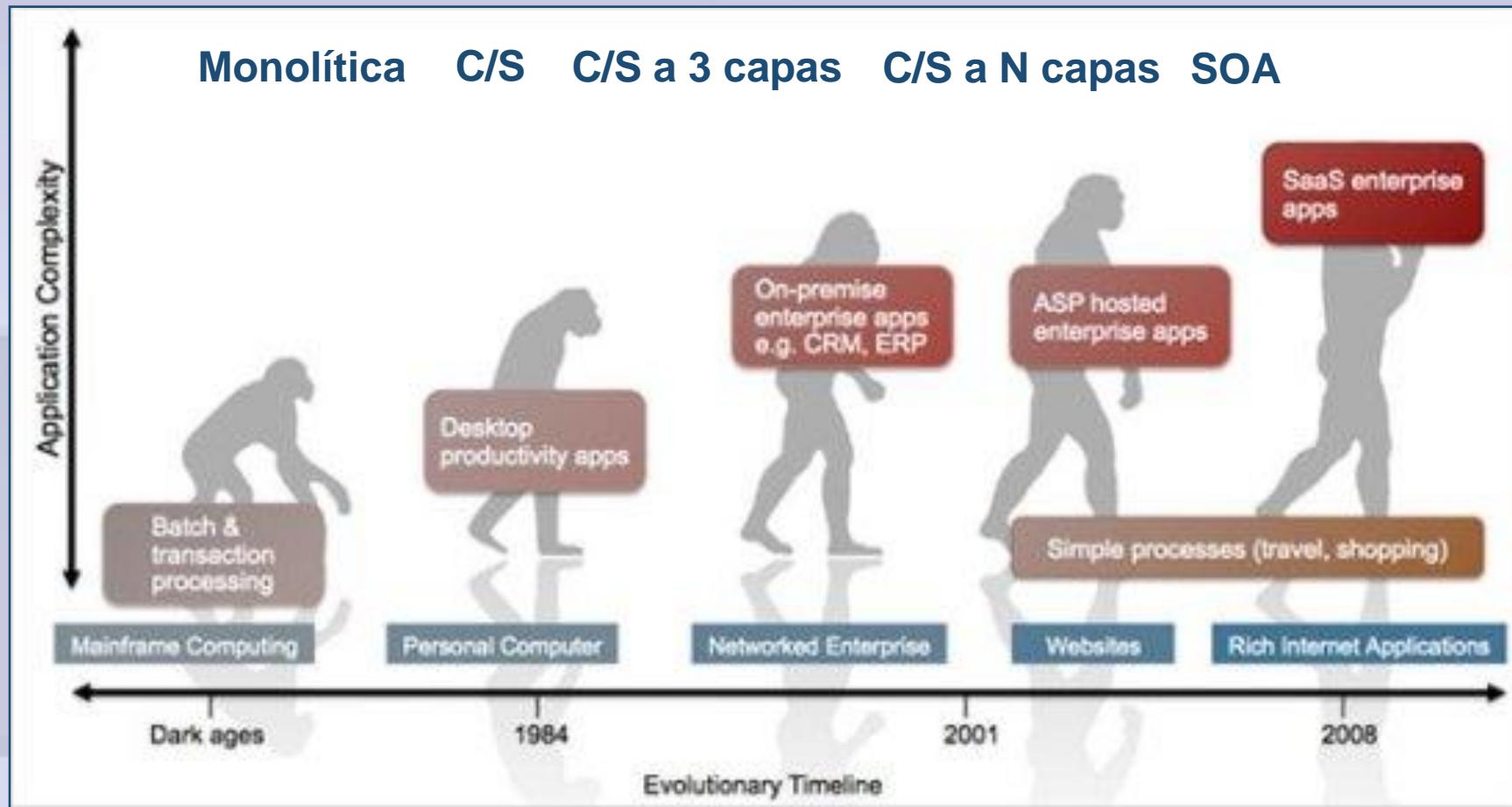
The technology adoption lifecycle: Everett Rogers Diffusion of Innovations - 1957



Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. **Evolución de la Arquitectura de Software**
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

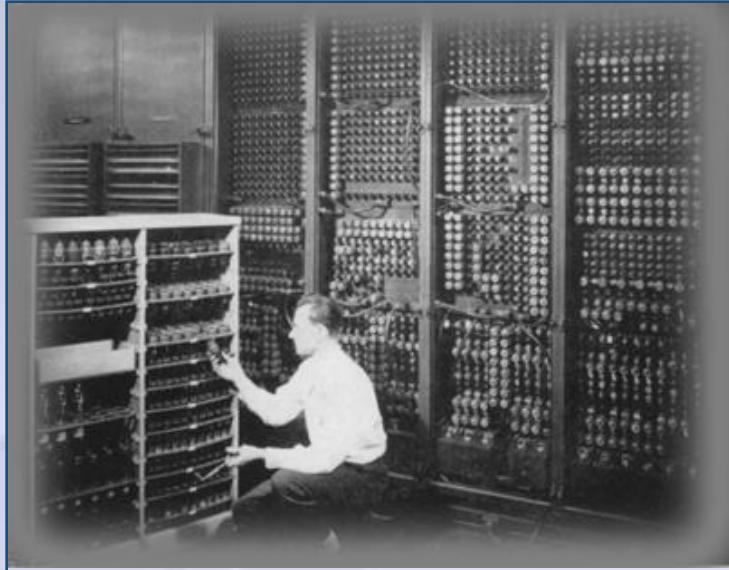
Evolución de la Arquitectura de Software



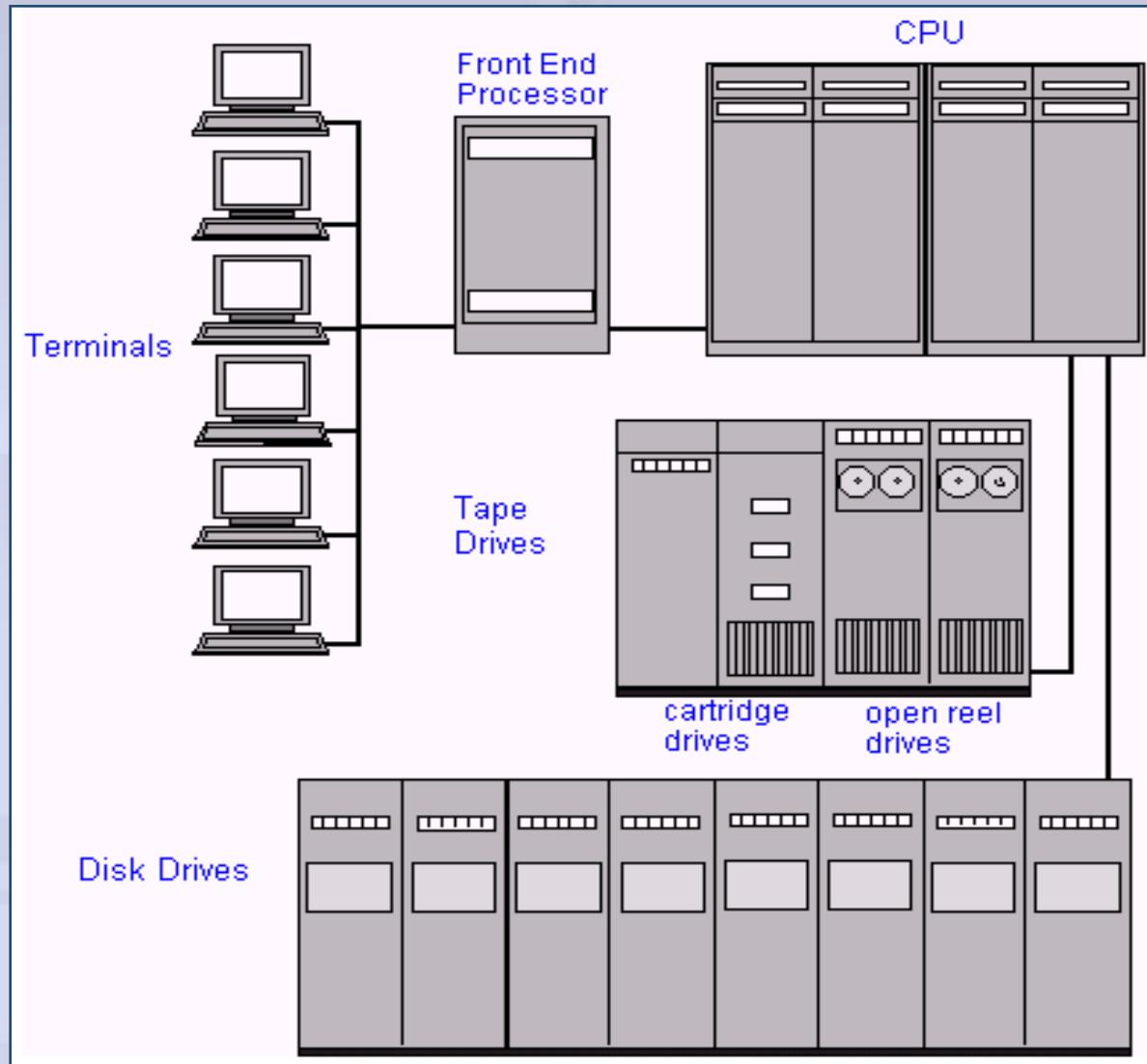
Arquitectura Monolíticas

- Mainframes de gran tamaño. En 1946 ENIAC
 - Superficie de 160 m²
 - Peso 30 toneladas,
 - Procesamiento de 30.000 instrucciones/seg.
- Sistemas operativos propietarios.
- Solo texto.
- Terminales “brutas” con teclado y pantalla por ejemplo de ámbar que se conectan físicamente a un Mainframe.

Arquitectura Monolíticas



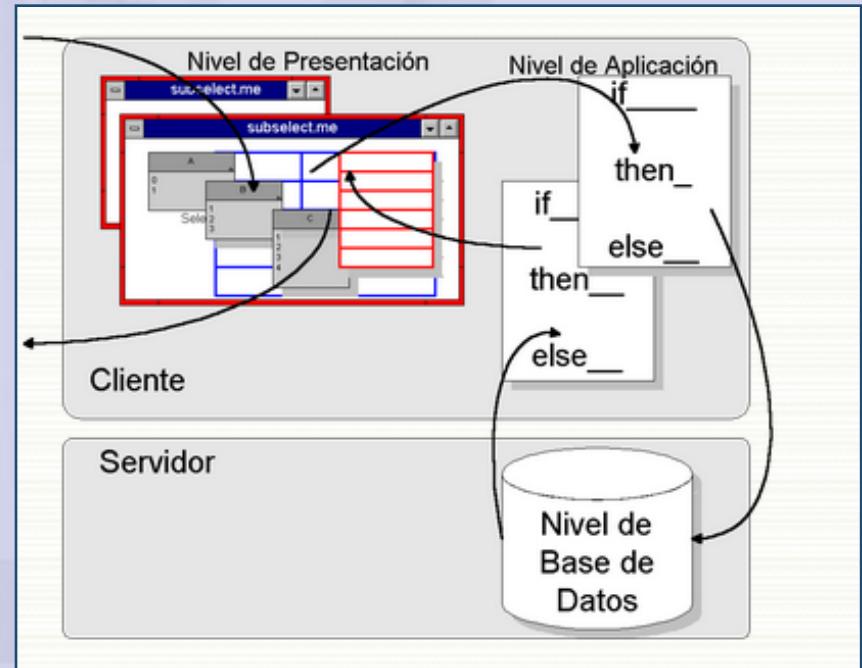
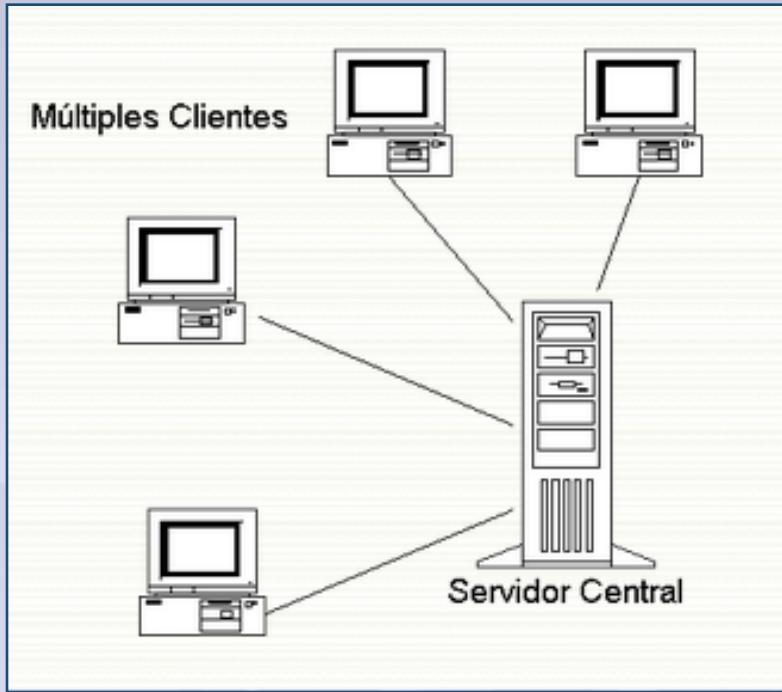
Arquitectura Monolíticas



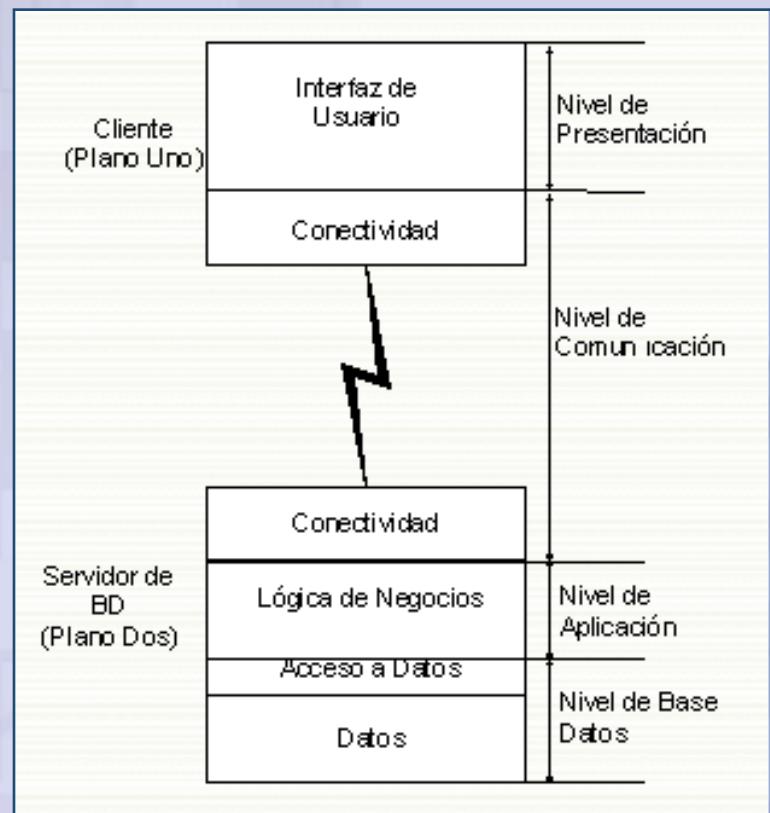
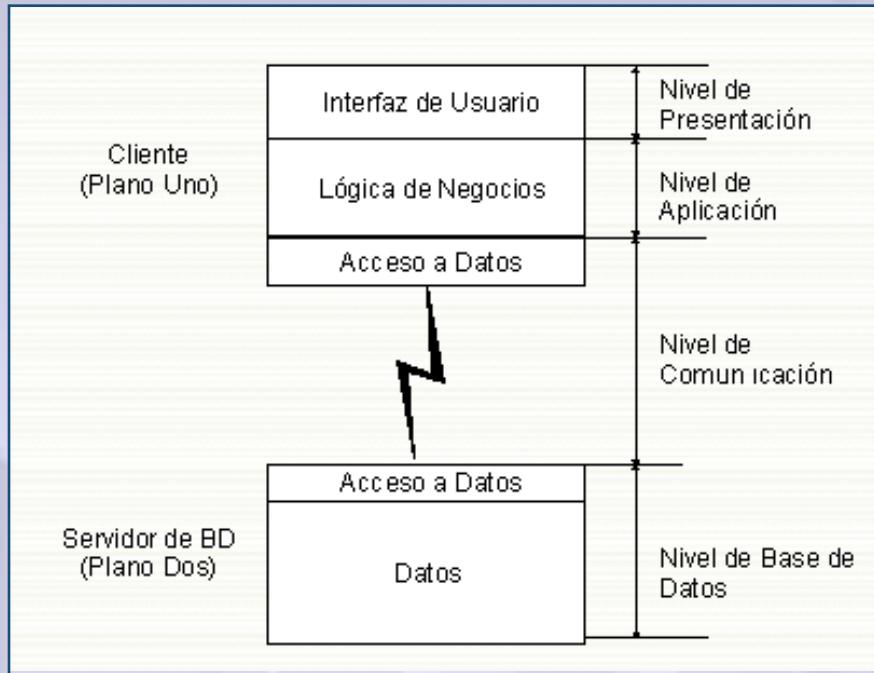
C/S a Dos Capas

- Disminución del tamaño de los Mainframes que empezaron a ser llamados Server.
- Aparición de Windows.
- Fortalecimiento de las redes LAN.
- Conviven diferentes sistemas operativos:
 - Los servidores usaban frecuentemente Unix.
 - Los clientes usaban frecuentemente Windows.

C/S a Dos Capas

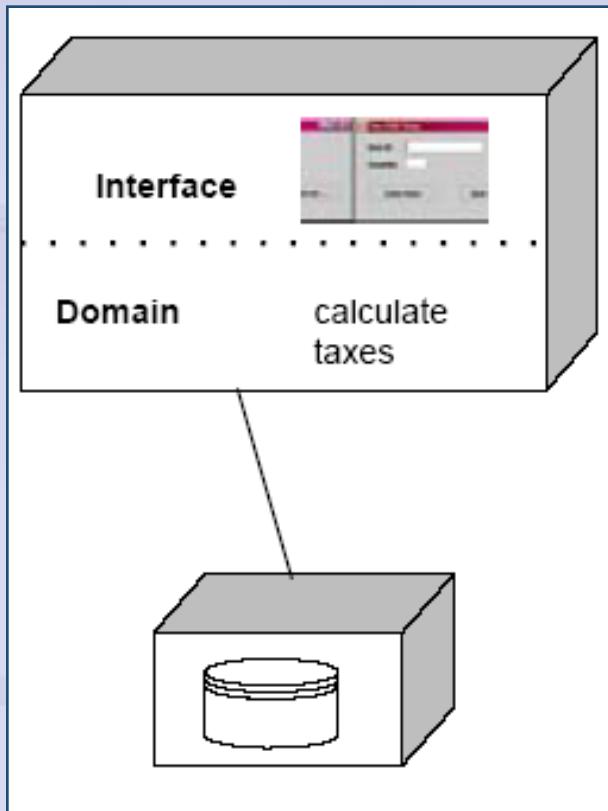


C/S a Dos Capas

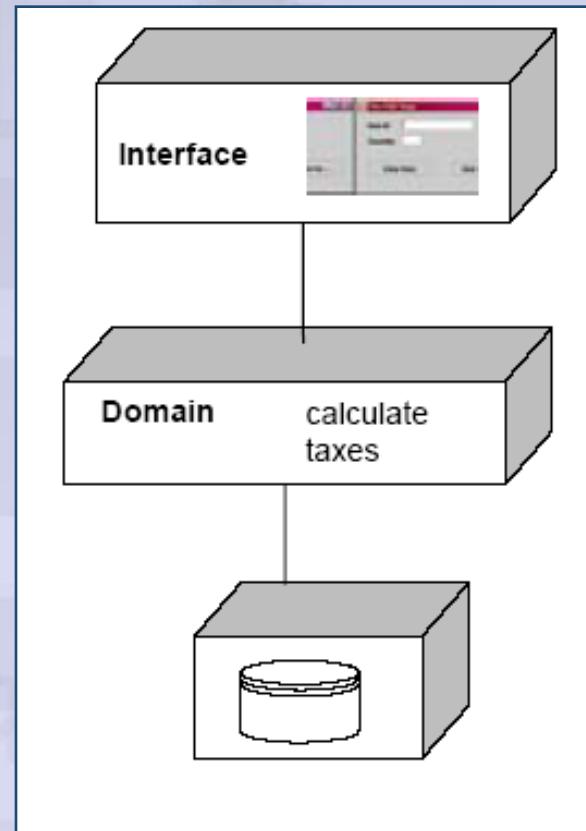


C/S a Dos Capas (Thin Client vs. Fat Client)

Cliente Grueso (Denso) vs. Cliente Fino (Delgado)



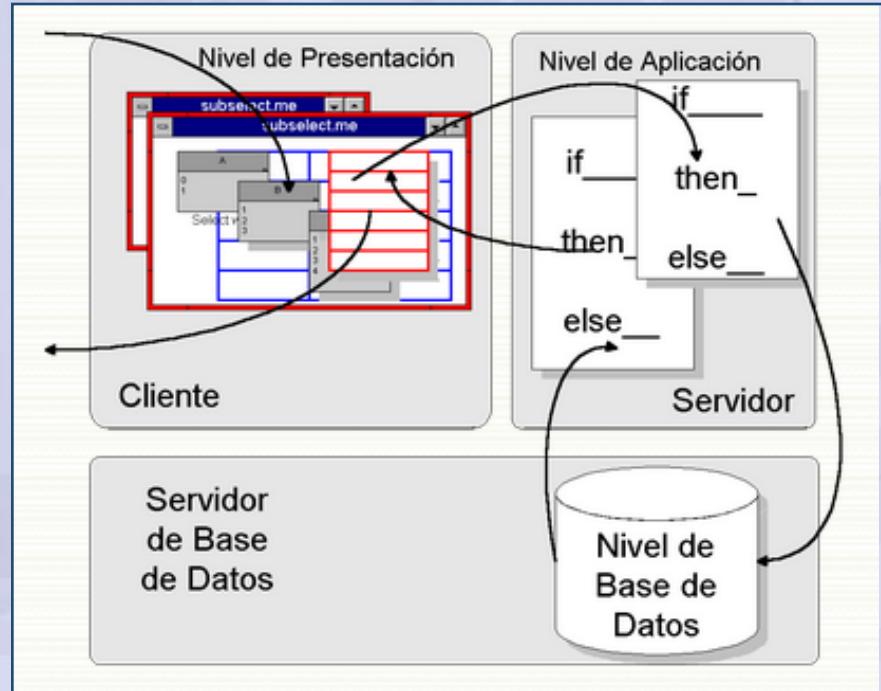
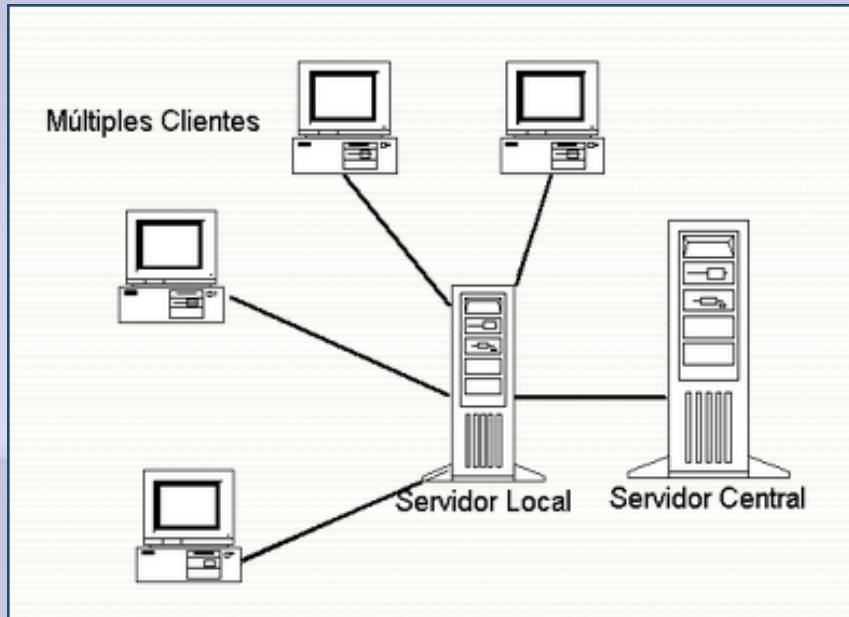
VS.



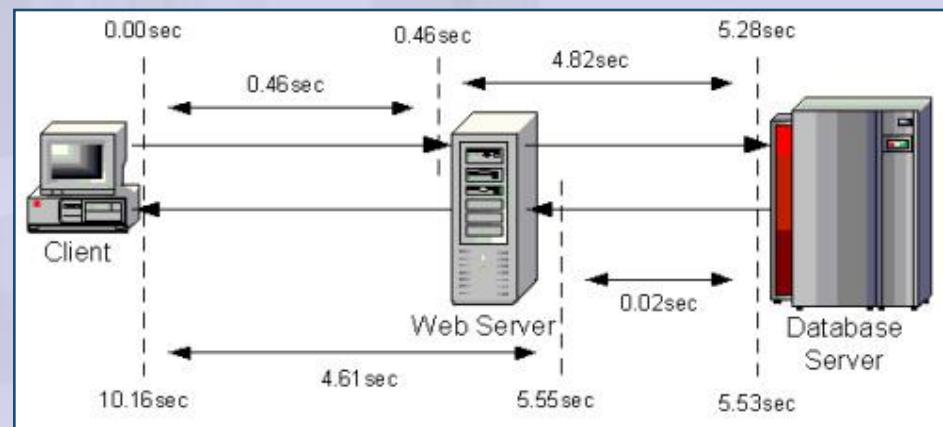
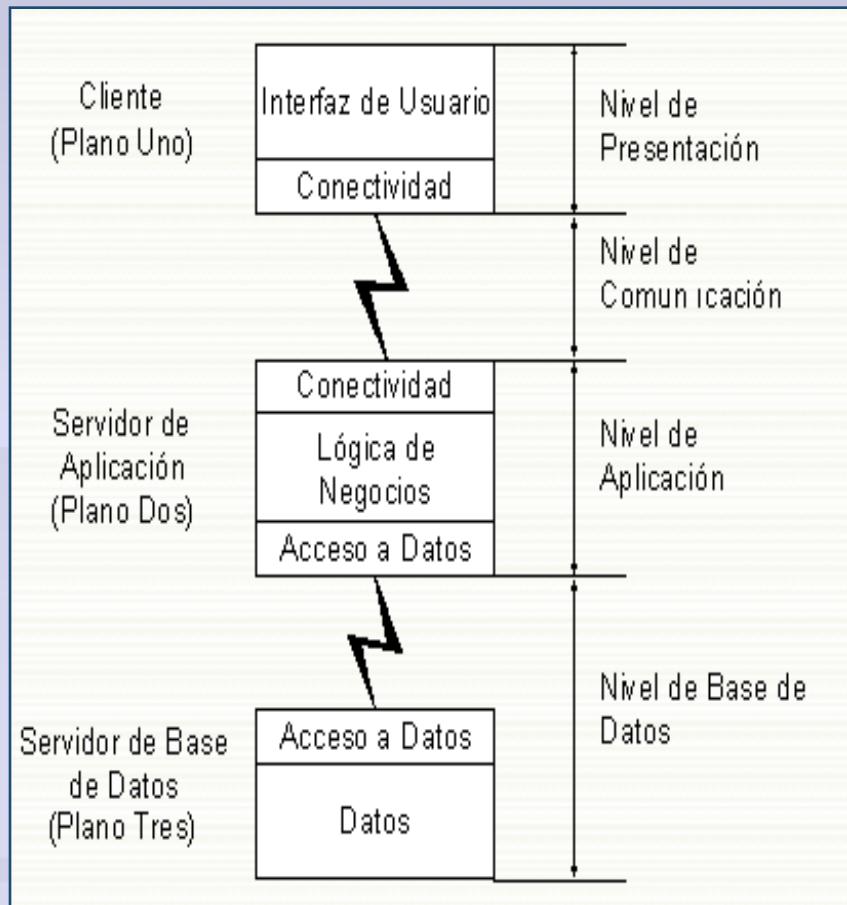
C/S a Tres Capas

- Disminución del costo y masificación de los computadores. En 2002 un Pentium IV a 2 Ghz.
 - Superficie de 217 mm²
 - 5.300 MTOPS (“Millions of theoretical operations per second)
- Aparición comercial de Internet.
- Fortalecimiento de las redes WAN.
- Conviven diferentes:
 - Tipos de maquinas.
 - Sistemas operativos.
 - Protocolos de comunicaciones.

C/S a Tres Capas

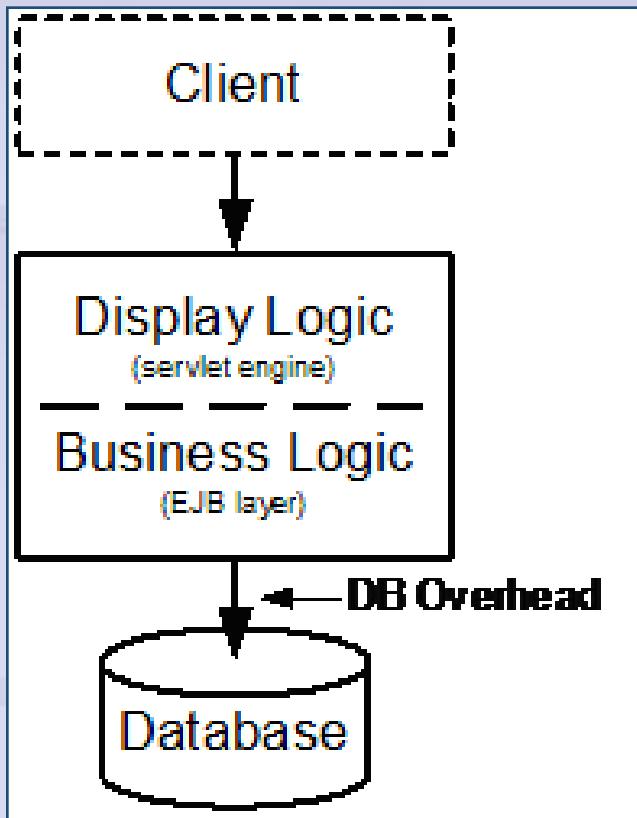


C/S a Tres Capas

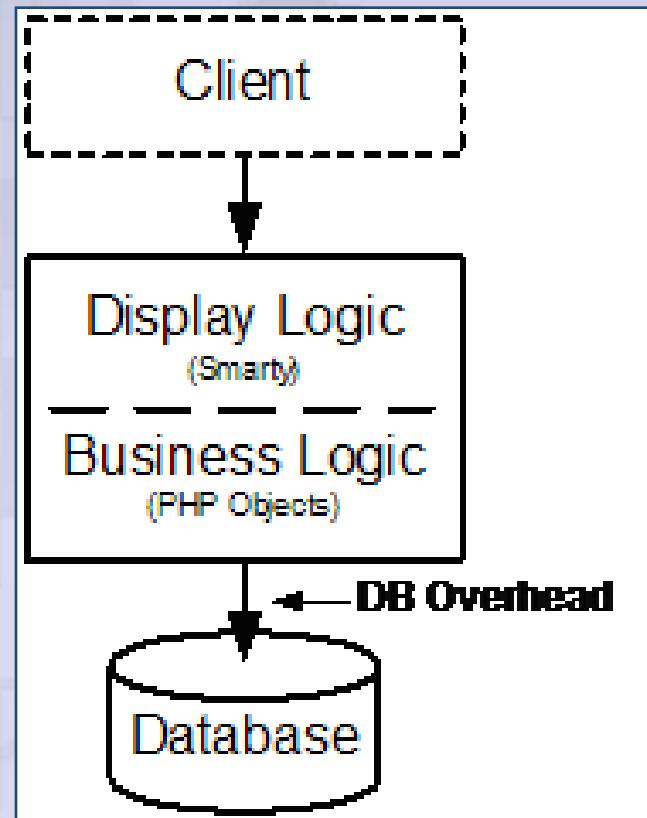


C/S a Tres Capas

- Interpretación de la Arquitectura J2EE en EJB 2.0

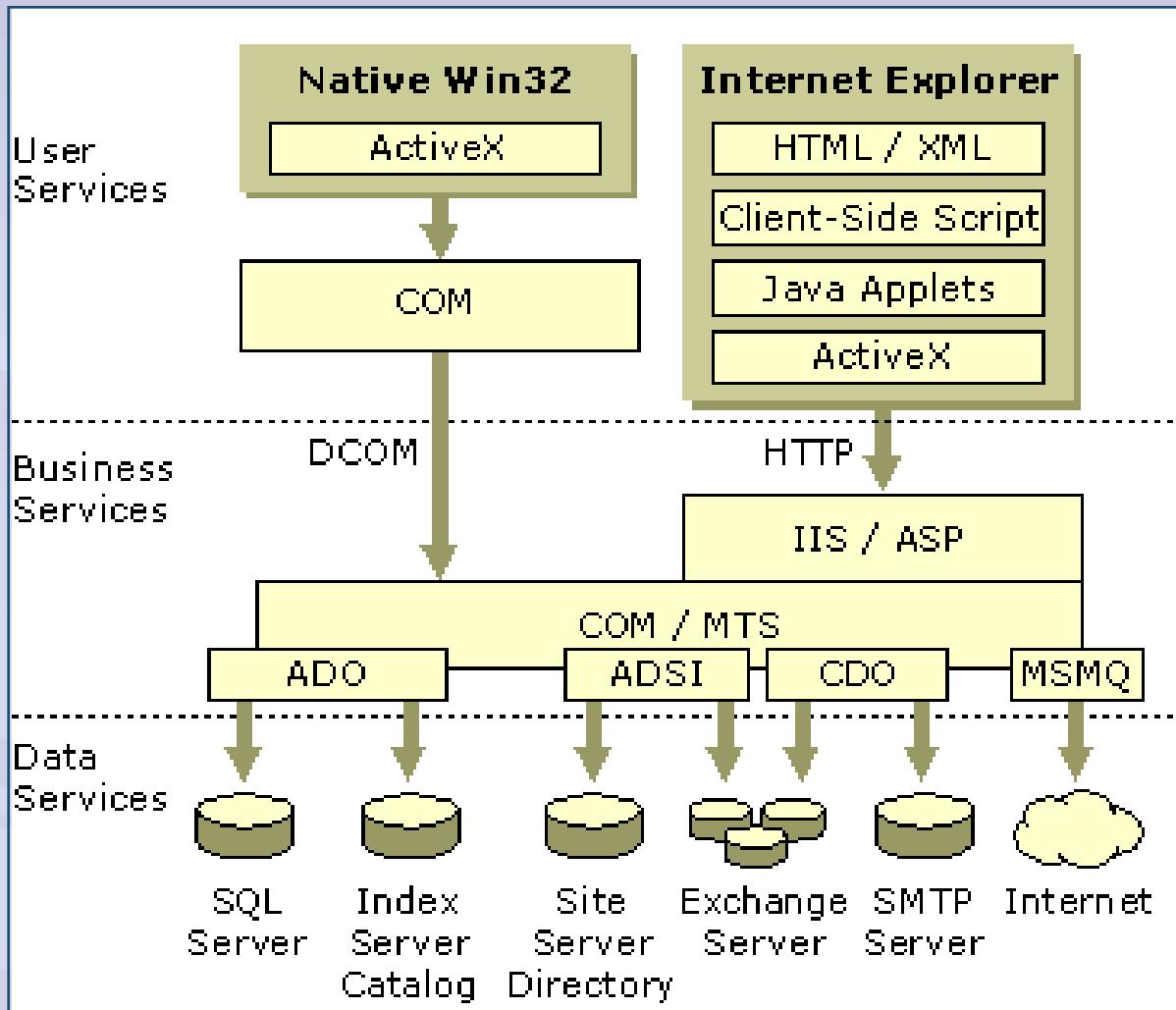


- Artificio de escalabilidad en PHP similar a EJB 2.0



C/S a Tres Capas

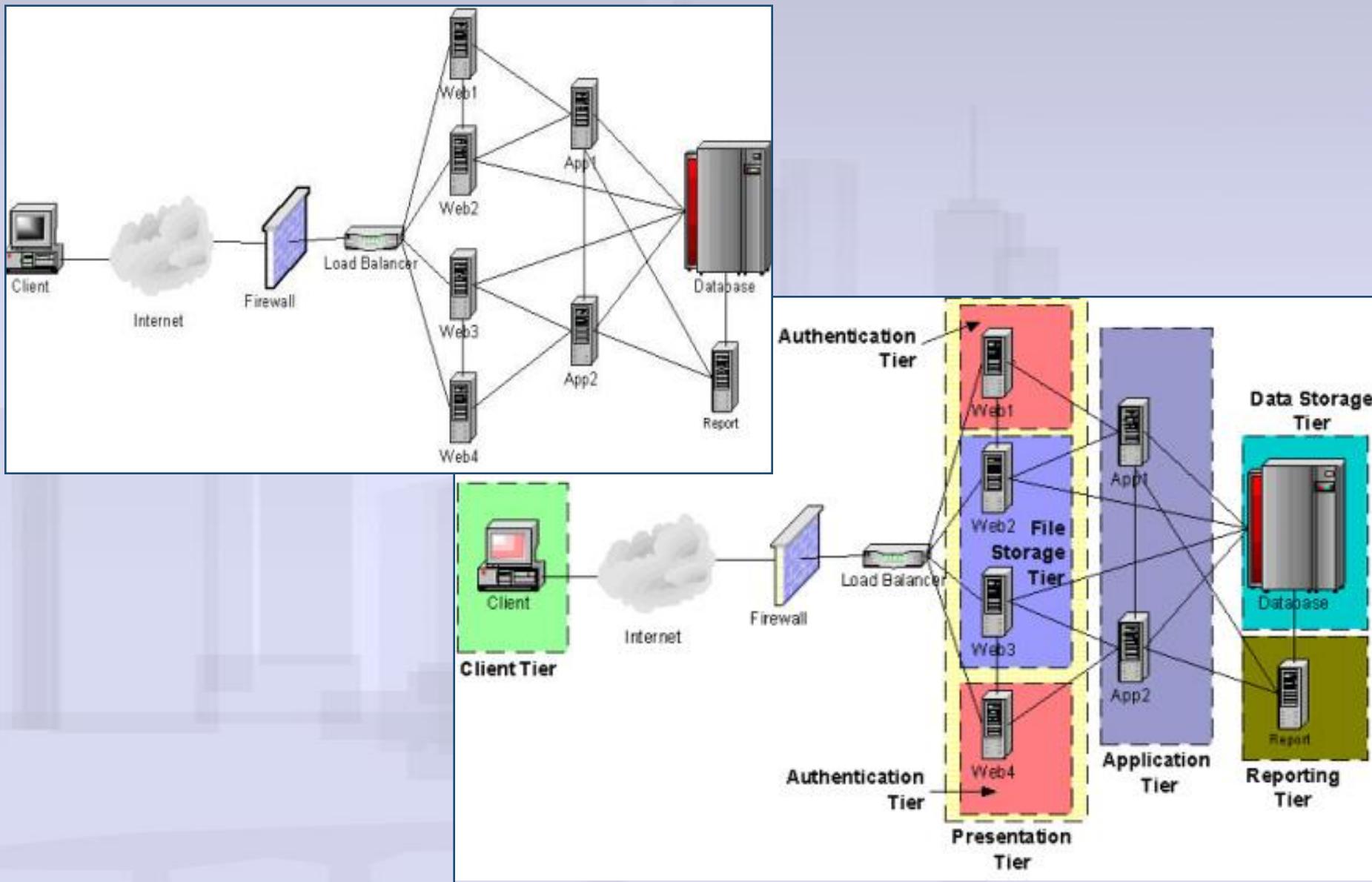
Microsoft Technologies for Three-Tier Applications



C/S a N Capas

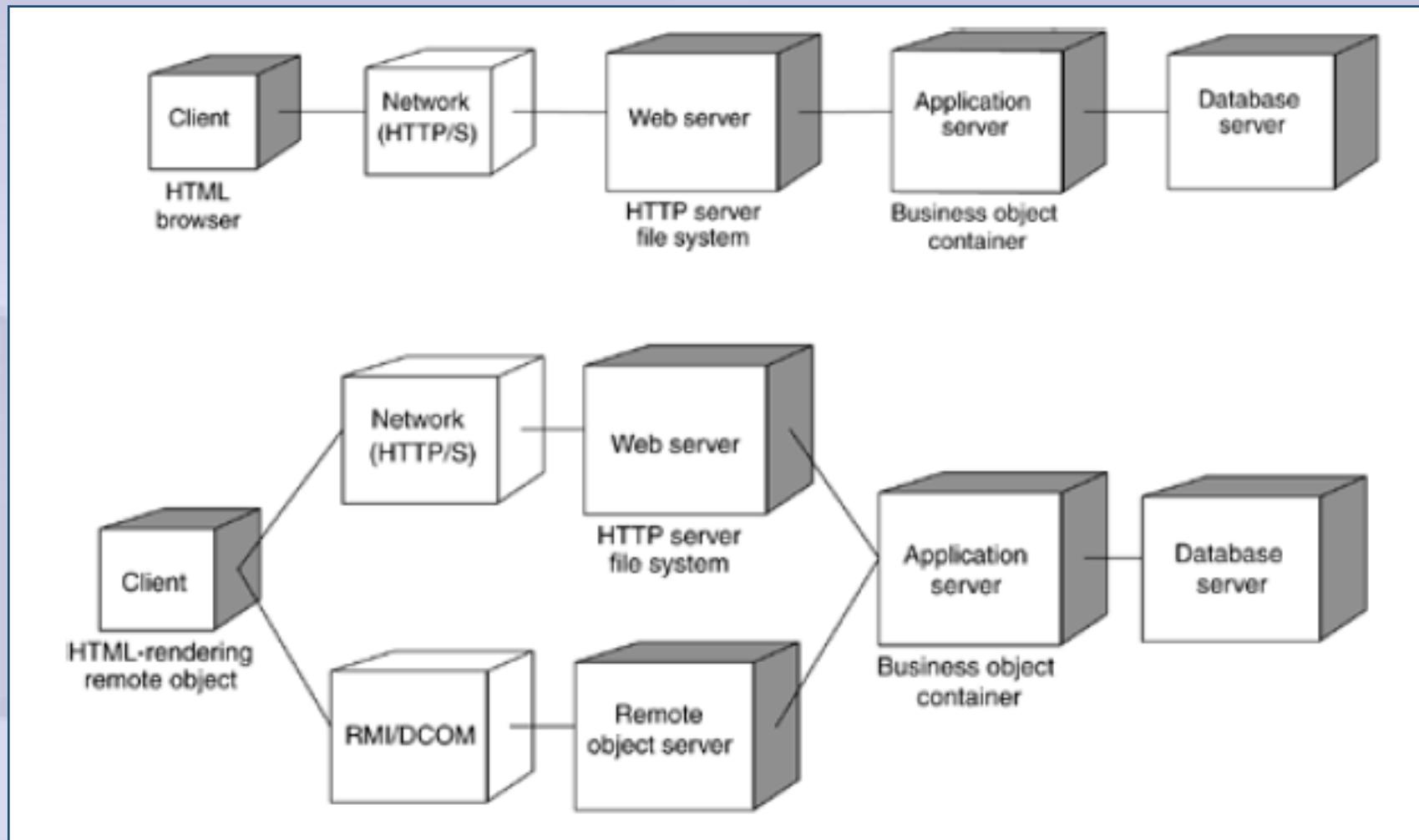
- Masificación de Internet y de las comunicaciones inalámbrica.
- Aparición dispositivos móviles para ejecutar aplicaciones.
- Integración de negocios e interoperabilidad.
- Conviven diferentes:
 - Plataformas.
 - Middleware.
 - Arquitecturas.

C/S a N Capas

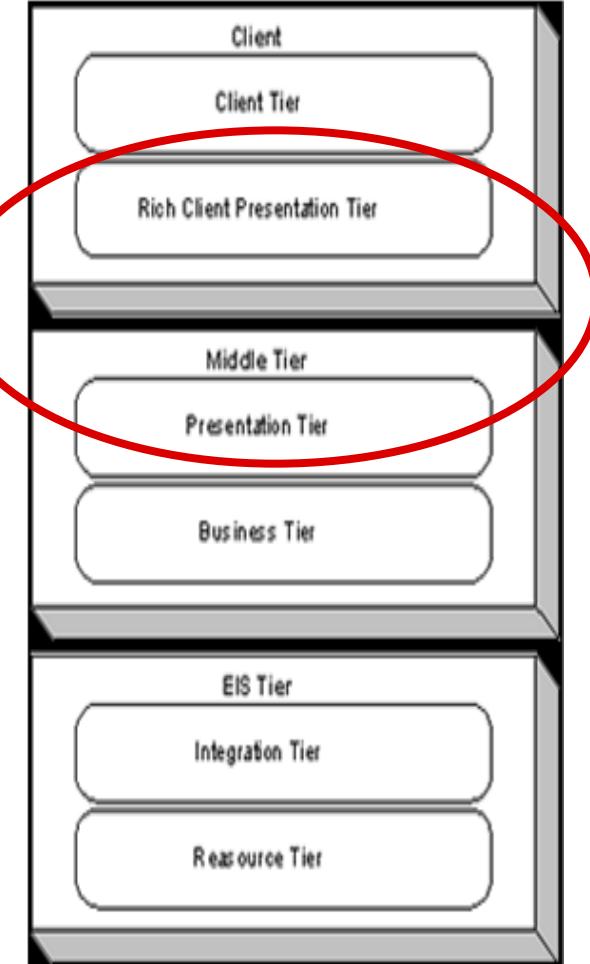
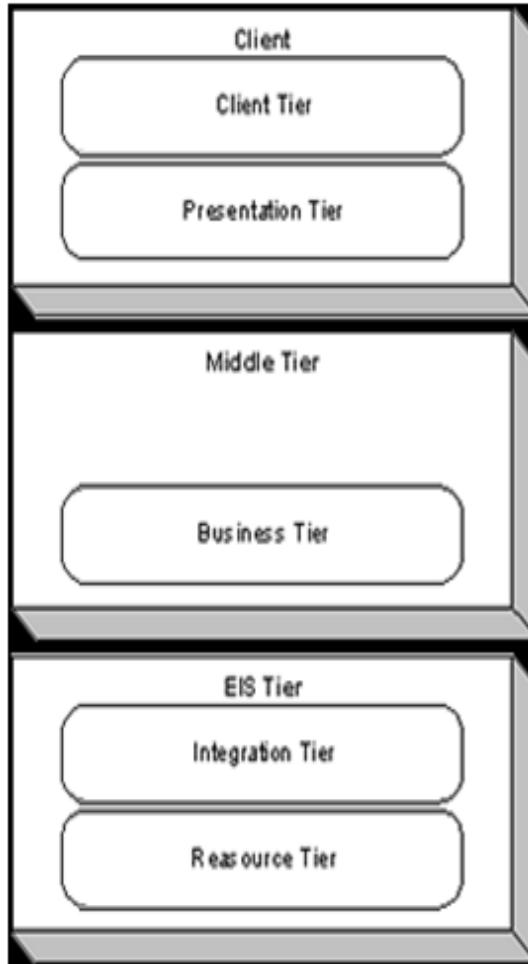
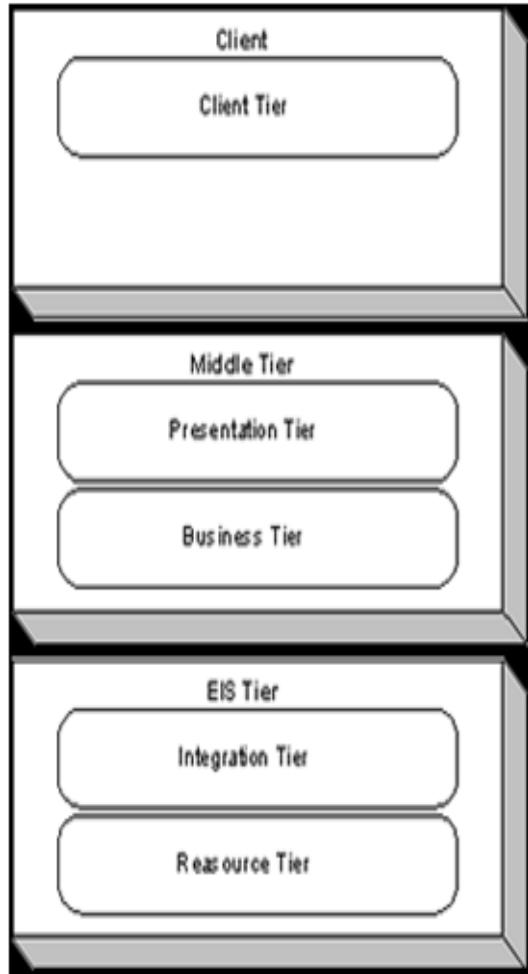


C/S a N Capas

HTTP con clientes Thin/Fat

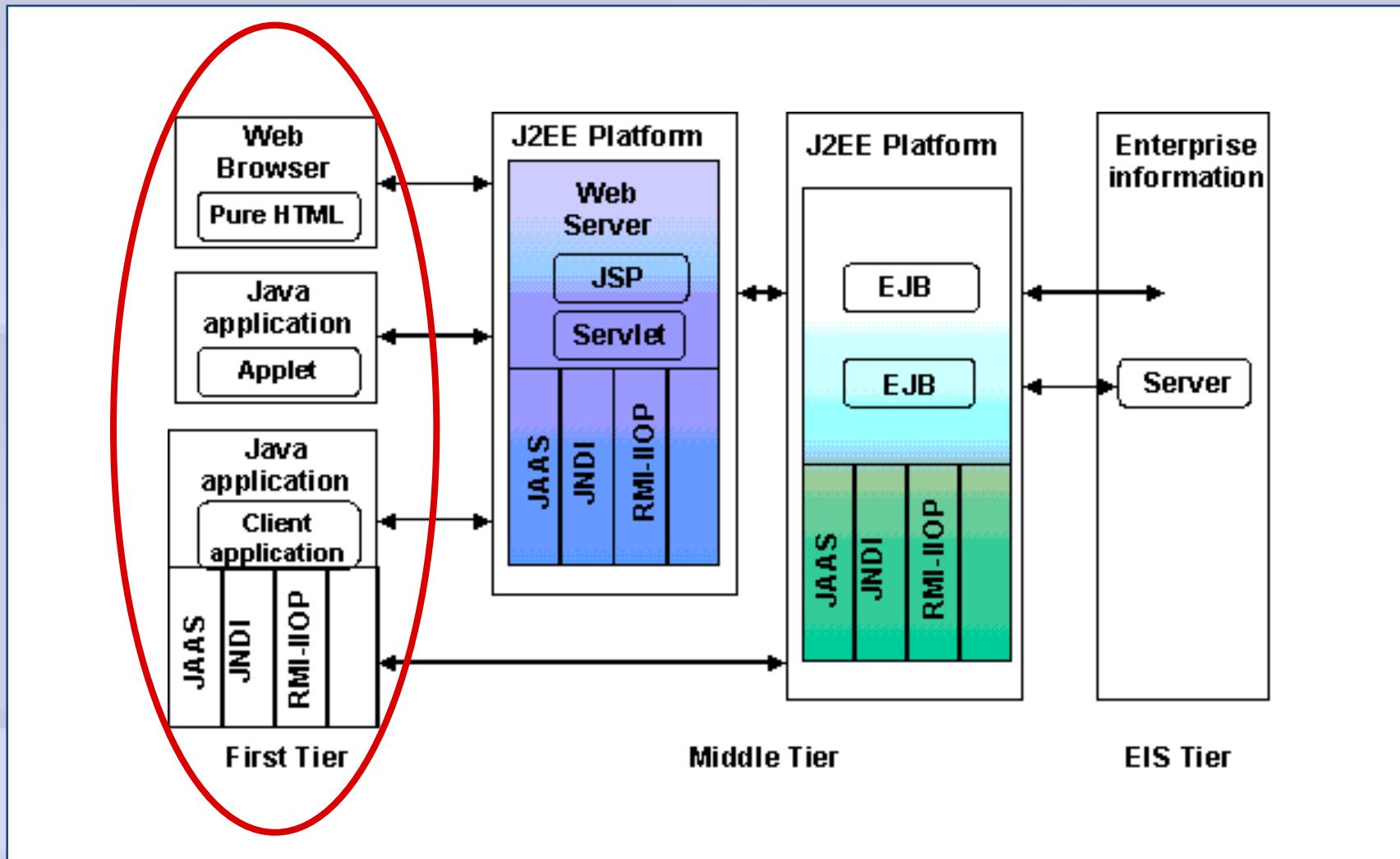


C/S a N Capas con RIA



C/S a N Capas

Tipos de Clientes en Java



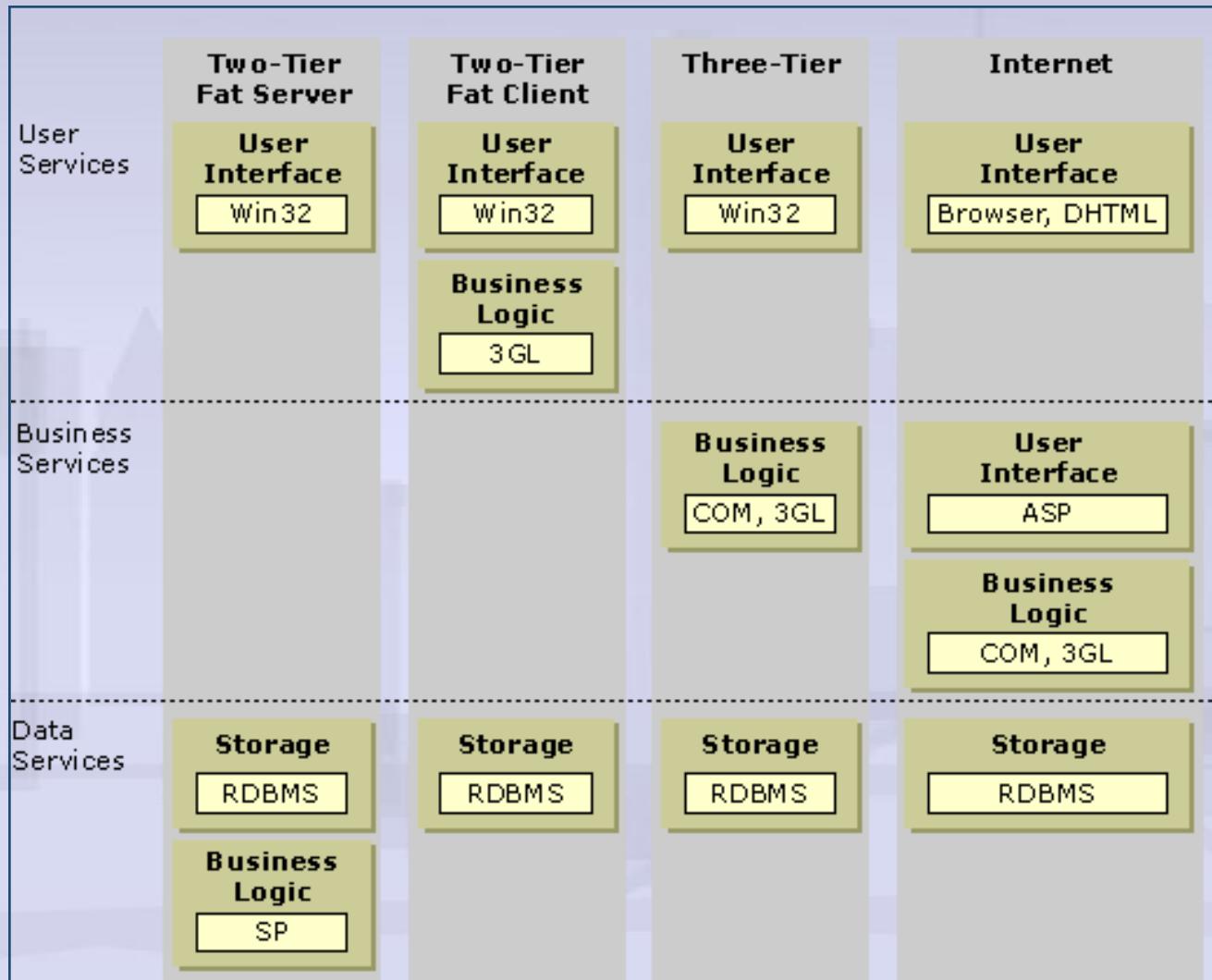
C/S a N Capas

J2EE Web Server Architecture

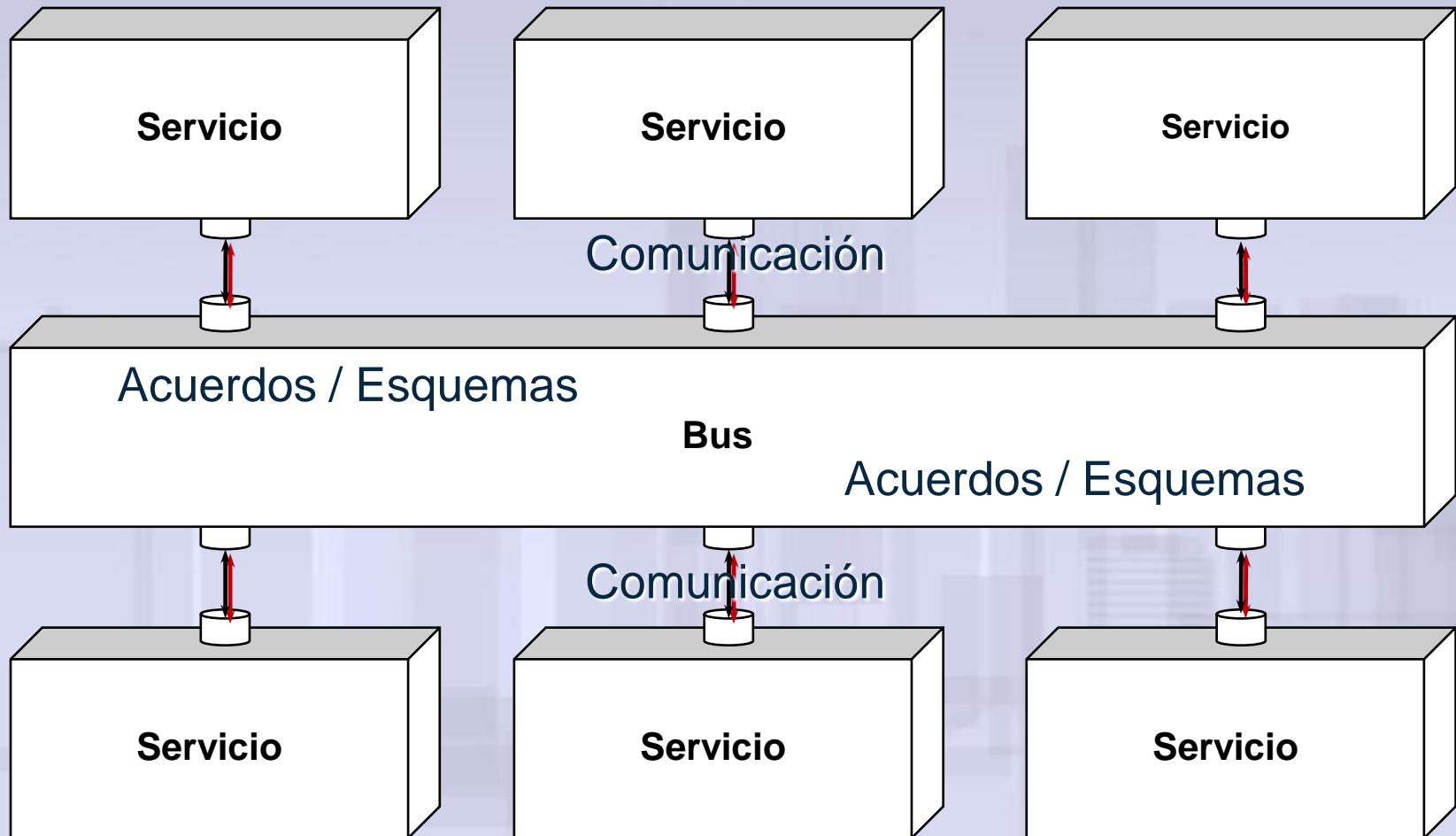
Nombre	Quiénes la componen	Dónde se ubica
Capa Cliente	Aplicaciones cliente, applets, aplicaciones y otras GUIs	PC Cliente
Capa de Presentación	JSP, Servlet y otras UIs	Servidor J2EE
Capa de Negocios	EJBs y otros objetos de negocios	Servidor J2EE
Capa de Integración	JMS, JDBC	Servidor J2EE
Capa de Recursos	Bases de Datos, Sistemas Externos	Servidor BD

C/S a N Capas (From Two-Tier to N-Tier)

Multitier Architecture



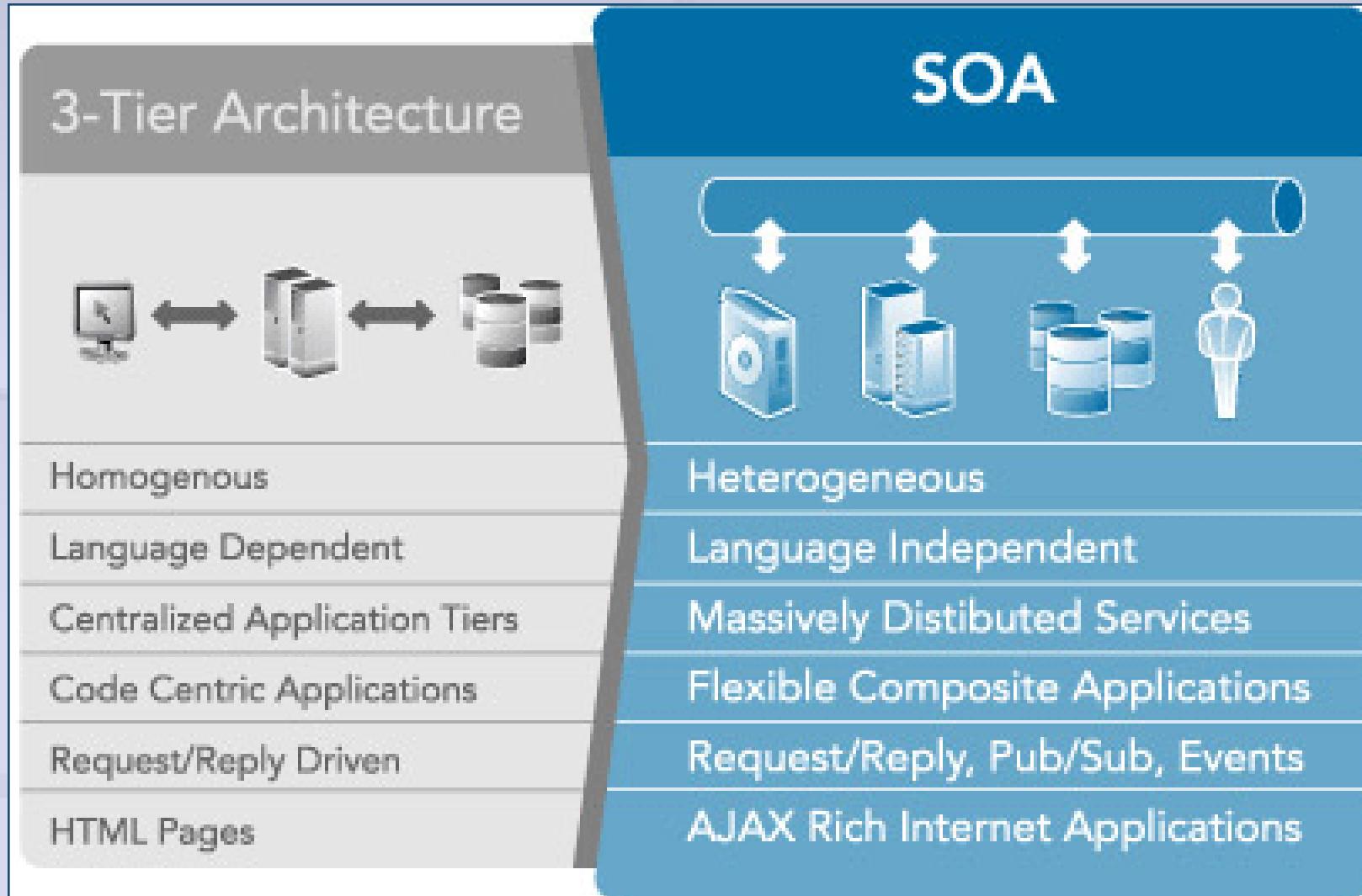
Arquitectura Orientada a Servicios



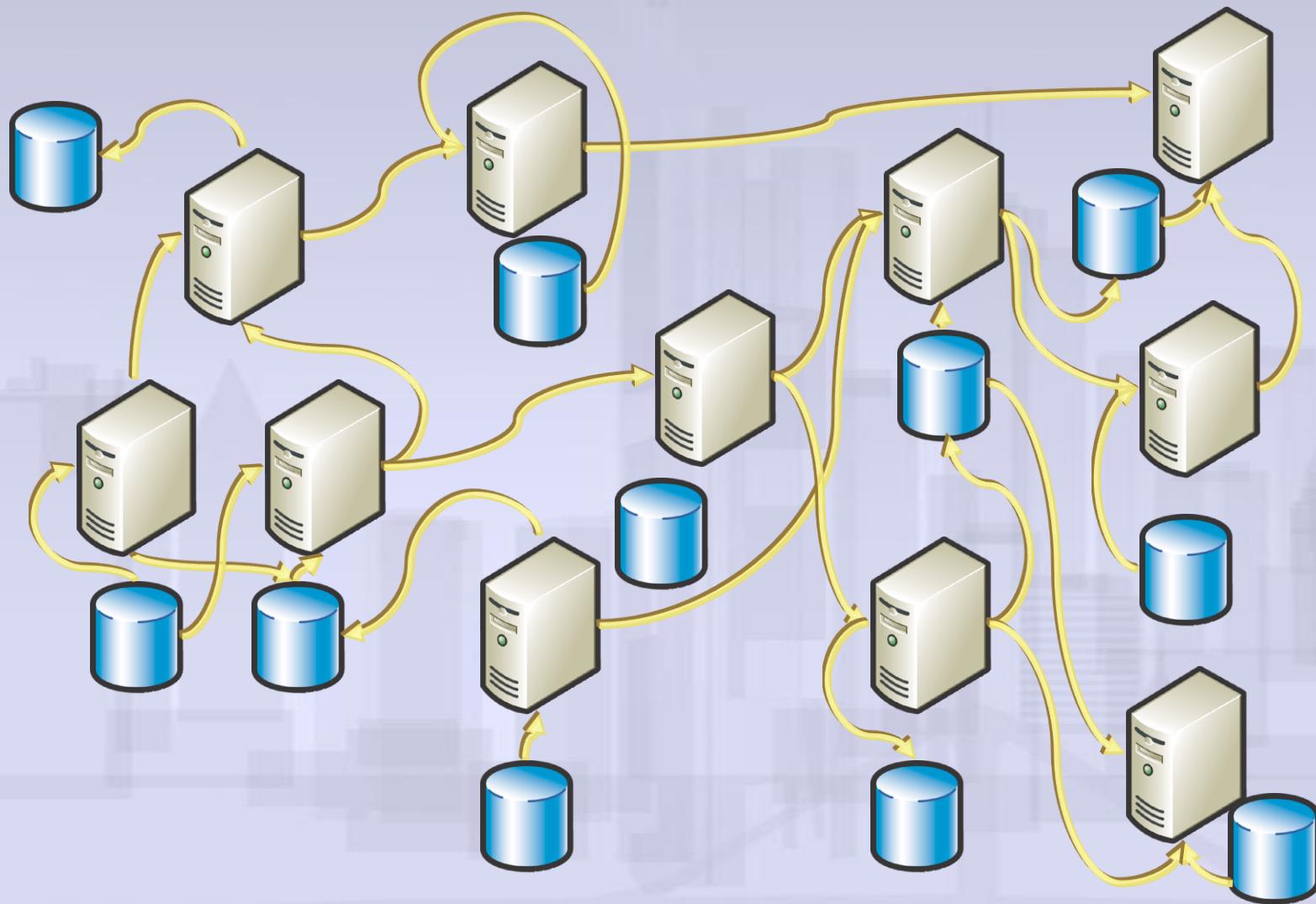
Arquitectura Orientada a Servicios

- Una aproximación para construir sistemas usando servicios los cuales se adhieren a 4 pilares:
 - Los límites son explícitos
 - Los servicios son Autónomos
 - Los servicios comparten esquemas y contratos, no clases
 - La compatibilidad de los servicios, se determina basados en las políticas

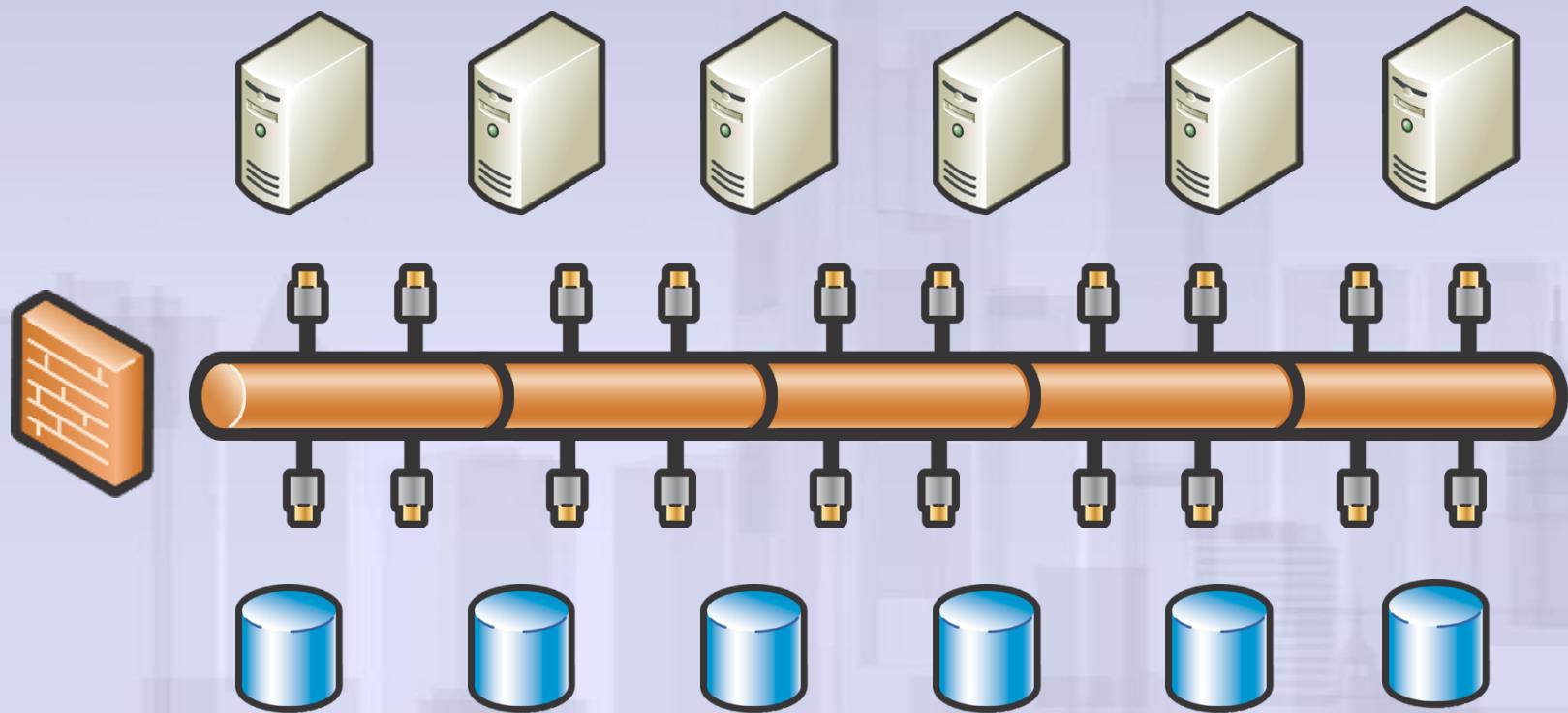
SOA: Beneficios de TI



Arquitectura Hoy en Día



Propuesta SOA



Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. **El rol del Arquitecto de Software**
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

Competencias del Arquitecto

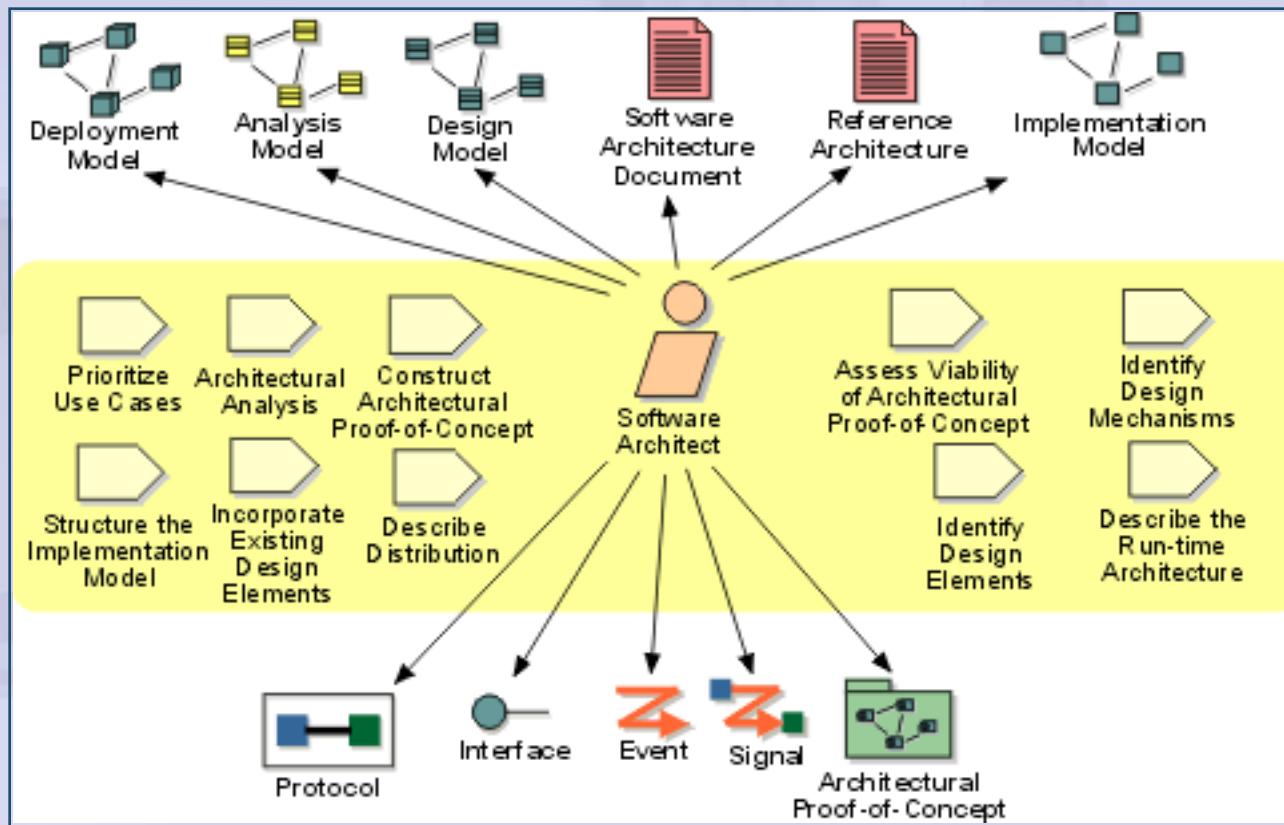
"The ideal architect should be a person of letters, a mathematician, familiar with historical studies, a diligent student of philosophy, acquainted with music, not ignorant of medicine, learned in the responses of jurisconsults, familiar with astronomy and astronomical calculations."

Vitruvius, circa 25 BC



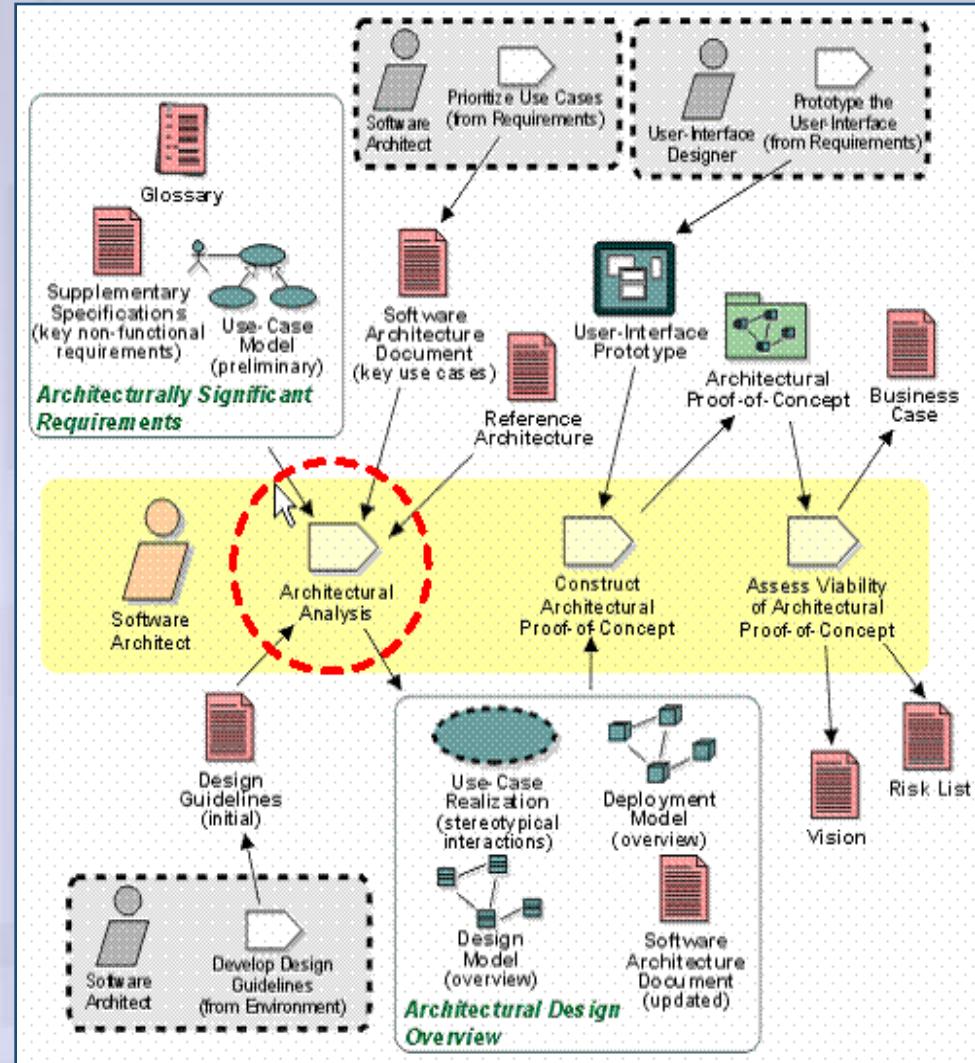
El Arquitecto de Software según RUP

The software architect role is responsible for the software architecture, which includes the key technical decisions that constrain the overall design and implementation for the project.

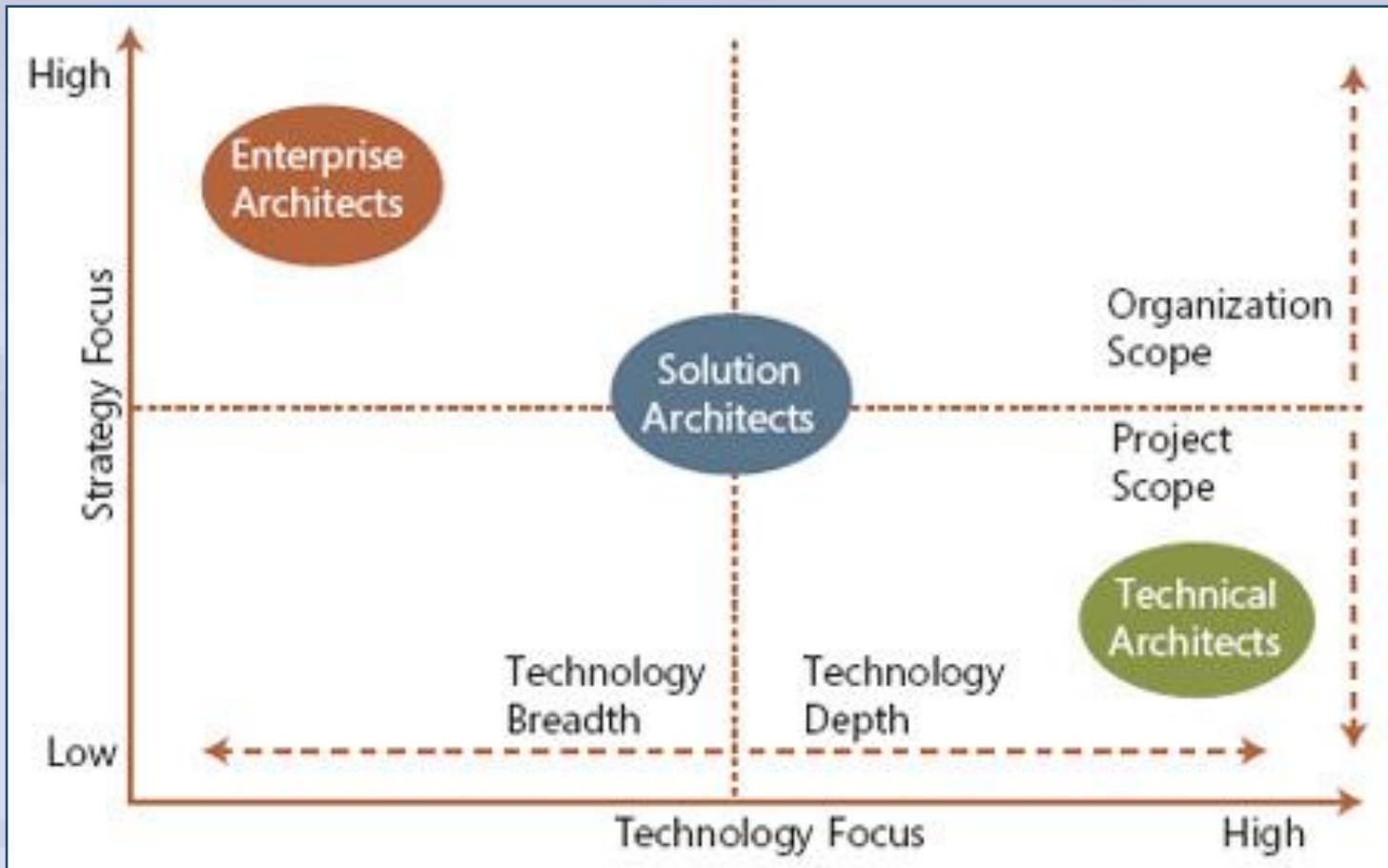


El Arquitecto de Software según RUP

- The software architect has overall responsibility for driving the major technical decisions, expressed as the software architecture. This typically includes identifying and documenting the architecturally significant aspects of the system, including requirements, design, implementation, and deployment "views" of the system.
- The architect is also responsible for providing rationale for these decisions, balancing the concerns of the various stakeholders, driving down technical risks, and ensuring that decisions are effectively communicated, validated, and adhered to.



Tipos de Arquitectos



Preocupaciones del Arquitecto



Responsabilidades del Arquitecto

Architecture

Architecture definition, system structure, logical view, physical view, architectural principles, security, etc.

Contributed To | Defined

Non-functional Requirements

Performance, scalability, security, etc.

Delivered Against | Justified | Tested

Project Methodology

Project structure and use of methodology such as waterfall, RUP, XP, Scrum, etc.

Contributed To | Defined

Hands-on Design, Development & Testing

UML diagrams, code, unit tests, etc.

Yes | No

Software Selection

Application stack, databases, libraries, frameworks, technology standards, etc.

Greenfield Project | Existing System

Infrastructure Selection

Operating systems, hardware, networks, disaster recovery, etc.

Greenfield Project | Existing System

Leadership

Technical leadership, responsibility and authority, steering the team, etc.

Contributed To | Performed

Coaching and Mentoring

Helping people with technical problems, helping people move into new roles, etc.

Design and Code | Architecture

Development Processes

Source code control, build process, continuous integration, automated testing and other development processes/tools.

Contributed To | Defined

Practices and Standards

Coding standards and guidelines, project practices, tool selection, etc.

Contributed To | Defined | Enforced

Breadth of Experience

Knowledge of many technologies and architectures.

Yes | No

Software Development And Technology Trends

Agile, Web 2.0, SOA, lightweight Java EE...

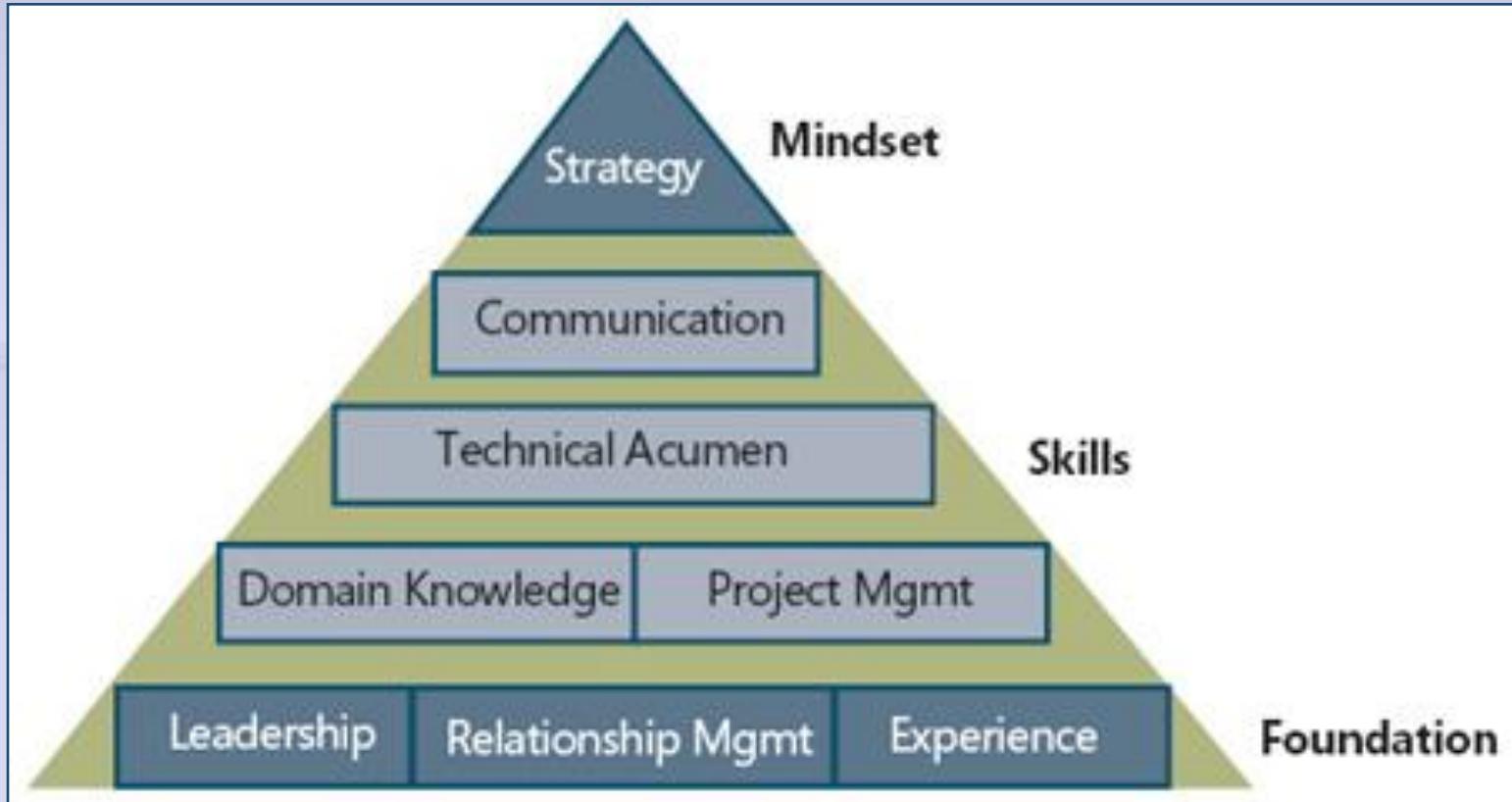
Awareness | Opinions



This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License.

Coding the Architecture (www.codingthearchitecture.com)

Competencias del Arquitecto de Software



Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. **Propuestas Arquitectónicas**
6. Conclusiones
7. Referencias

Propuestas Arquitectónicas

Zachman (Niveles)	TOGAF (Arquitecturas)	4+1(Vistas)	[BRJ99] (Vistas)	POSA(Vistas)	Microsoft (Vistas)
Scope	Negocios	Lógica	Diseño	Lógica	Lógica
Empresa	Datos	Proceso	Proceso	Proceso	Conceptual
Sistema lógico	Aplicación	Física	Implementación	Física	Física
Tecnología	Tecnología	Desarrollo	Despliegue	Desarrollo	
Representación		Casos de uso	Casos de uso		
Funcionamiento					

Propuesta de Zachman

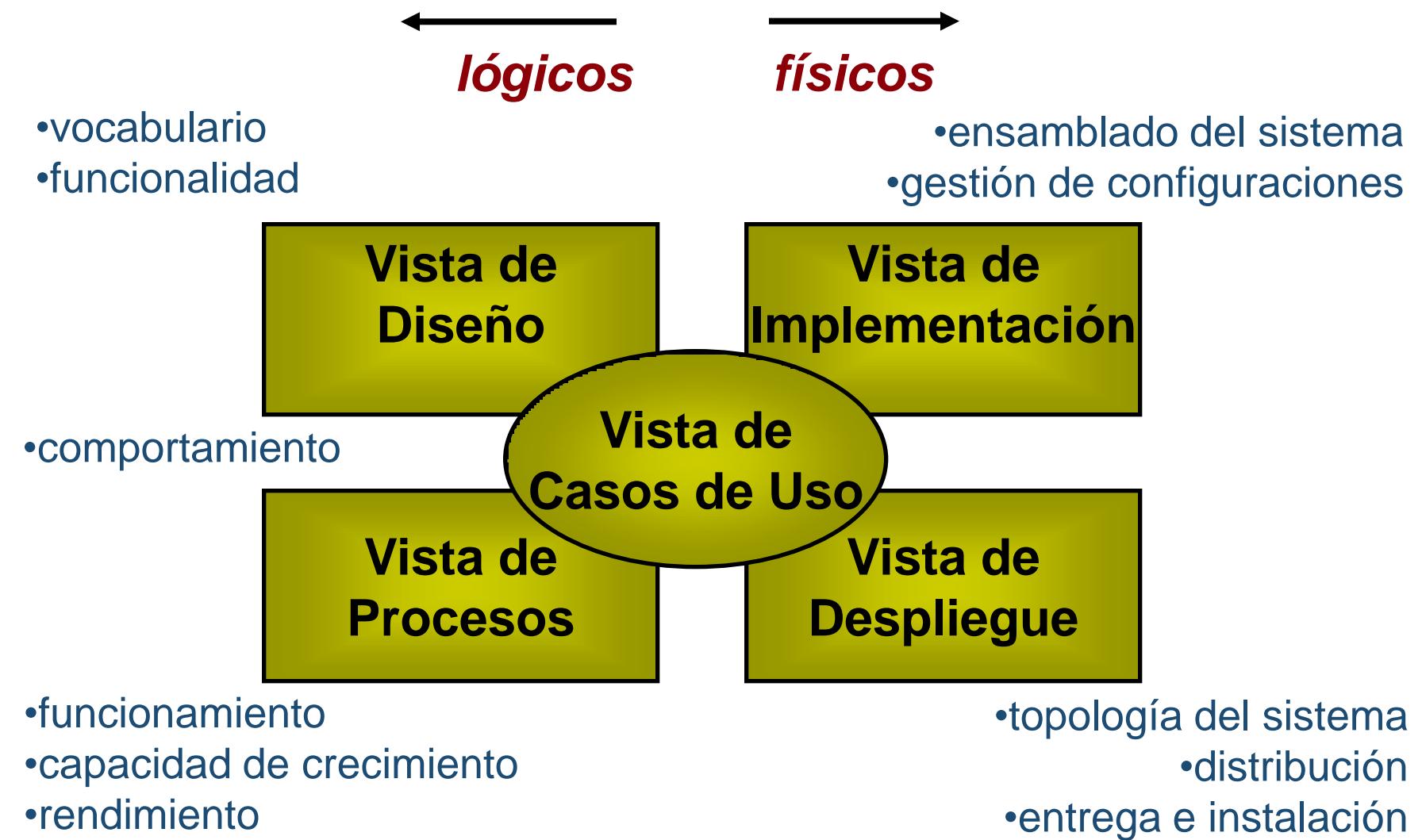
	Data (What)	Function (How)	Network (Where)	People (Who)	Time (When)	Motivation (Why)
Scope (Ballpark) view						
Owners View (Enterprise Model)						
Designers View (System Model)						
Builder's View (Technology Model)						
Out of Context View (Detailed Model)						
Operational View (Functioning)						

Propuesta de Zachman

	Data	Function	Network	People	Time	Motive
Planner's View	Business Things  Entity = Class of Business Thing	Processes Performed  Function = Class of Business Process	Business Locations  Node = Major Business Locations	Organizations  People = Major Organizations	Significant Events  Time = Major Business Event	Goals and Strategy  Ends/Means = Major Business Goals
Owner's View	Semantic Model  Ent = Business Entity Rel = Relationship	Process Model  Proc = Process I/O = Resources	Logistics System  Node = Location Link = Linkage	Work Flow Model  People = Organization Work = Work Product	Master Schedule  Time = Business Event Cycle = Business Cycle	Business Plan  End = Objective Means = Strategy
Designer's View	Logical Data Model  Ent = Data Entity Rel = Relationship	Application Architecture  Proc = Function I/O = User Views	System Architecture  Node = IS Function Link = Line Properties	Interface Architecture  People = Role Work = Deliverable	Processing Structure  Time = System Event Cycle = Processing	Business Rule Model  End = Structure Means = Action
Builder's View	Physical Data Model  Ent = Segment/Table Rel = Pointer/Key	System Design  Proc = Function I/O = Data Elements	Technology Architecture  Node = Hrdwr/Sftwr Link = Line Specs	Screen Architecture  People = User Work = Screen Format	Control Structure  Time = Execute Cycle = Component	Rule Design  End = Condition Means = Action
Integrator's View	Data Definition  Ent = Field Rel = Address	Program  Proc = Statement I/O = Control Block	Network Architecture  Node = Addresses Link = Protocols	Security Architecture  People = Identity Work = Job	Timing Definition  Time = Interrupt Cycle = Machine Cycle	Rule Design  End = Sub-Condition Means = Step
User's View	Data  Ent = Rel =	Function  Proc = I/O =	Network  Node = Link =	Organization  People = Work =	Schedule  Time = Cycle =	Strategy  End = Means =

Propuesta de Kruchten (4 + 1)

4+1 Vistas de Kruchten



Propuesta de Kruchten (4 + 1)

4+1 Vistas de Kruchten

Aspectos Estáticos:

- Diagramas de Clases
 - Diagramas de Objetos
 - Diagramas de Casos de Uso
 - Diagramas de Clases (clases activas)
 - Diagramas de Objetos (clases activas)
 - Diagramas de Componentes
 - Diagramas de Despliegue
-
- ```
graph TD; A[Vista de Diseño] --- B[Vista de Implementación]; A --- C((Vista de Casos de Uso)); A --- D[Vista de Procesos]; A --- E[Vista de Despliegue];
```

### Aspectos Dinámicos

- Diagramas de Interacción
- Diagramas de Estado
- Diagramas de Actividades

# Propuesta de Microsoft

## Perspectivas de la Arquitectura

### ■ Negocio:

Describe el funcionamiento interno del negocio central de la organización.

### ■ Aplicación:

Muestra las aplicaciones de la organización, su funcionalidad y relaciones.

### ■ Información:

Describe la información que maneja la organización y cómo está ligada a los circuitos de trabajo.

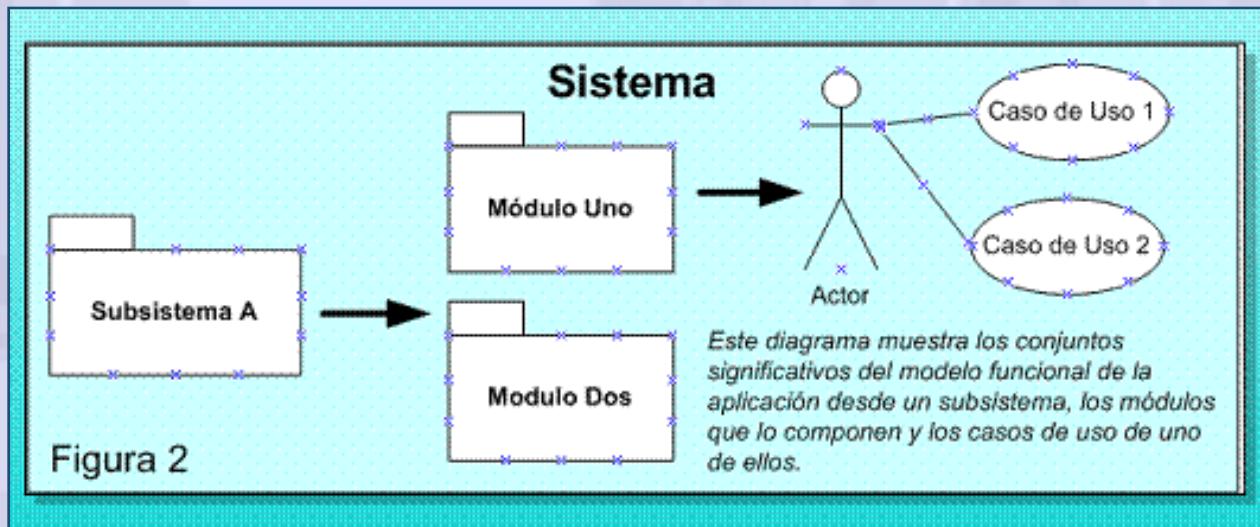
### ■ Tecnología:

Describe la estructura de hardware y software de base que da soporte informático a la organización.

# Propuesta de Microsoft

## Vista Conceptual

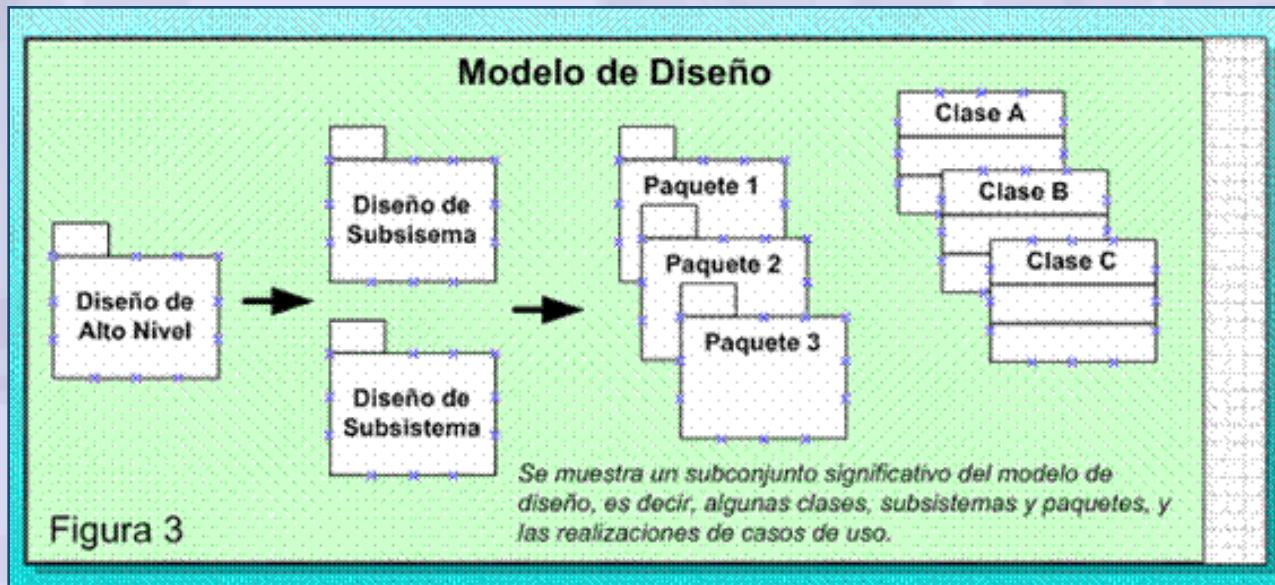
- Es usada para definir los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación y describir el modelo de negocio que la arquitectura debe cubrir. Esta vista muestra los subsistemas y módulos en los que se divide la aplicación y la funcionalidad que brinda dentro de cada uno de ellos.
- Casos de Uso, Diagramas de Actividad, Procesos de Negocio, Entidades del Negocio, etc.



# Propuesta de Microsoft

## Vista Lógica

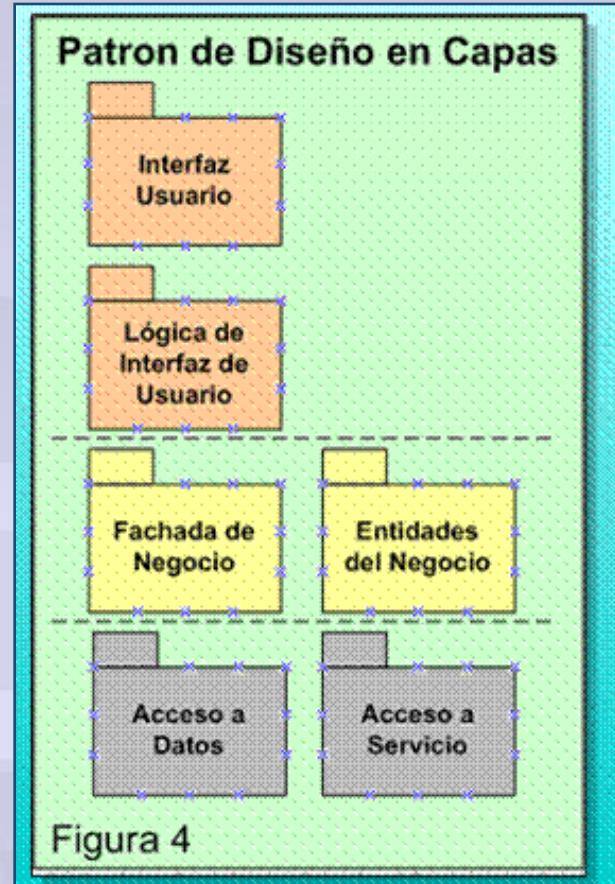
- Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución. Los arquitectos crean modelos de diseño de la aplicación, los cuales son vistas lógicas del modelo funcional y que describen la solución.
- Realización de los Casos de Uso, subsistemas, paquetes y clases de los casos de uso más significativos arquitectónicamente.



# Propuesta de Microsoft

## Vista Lógica

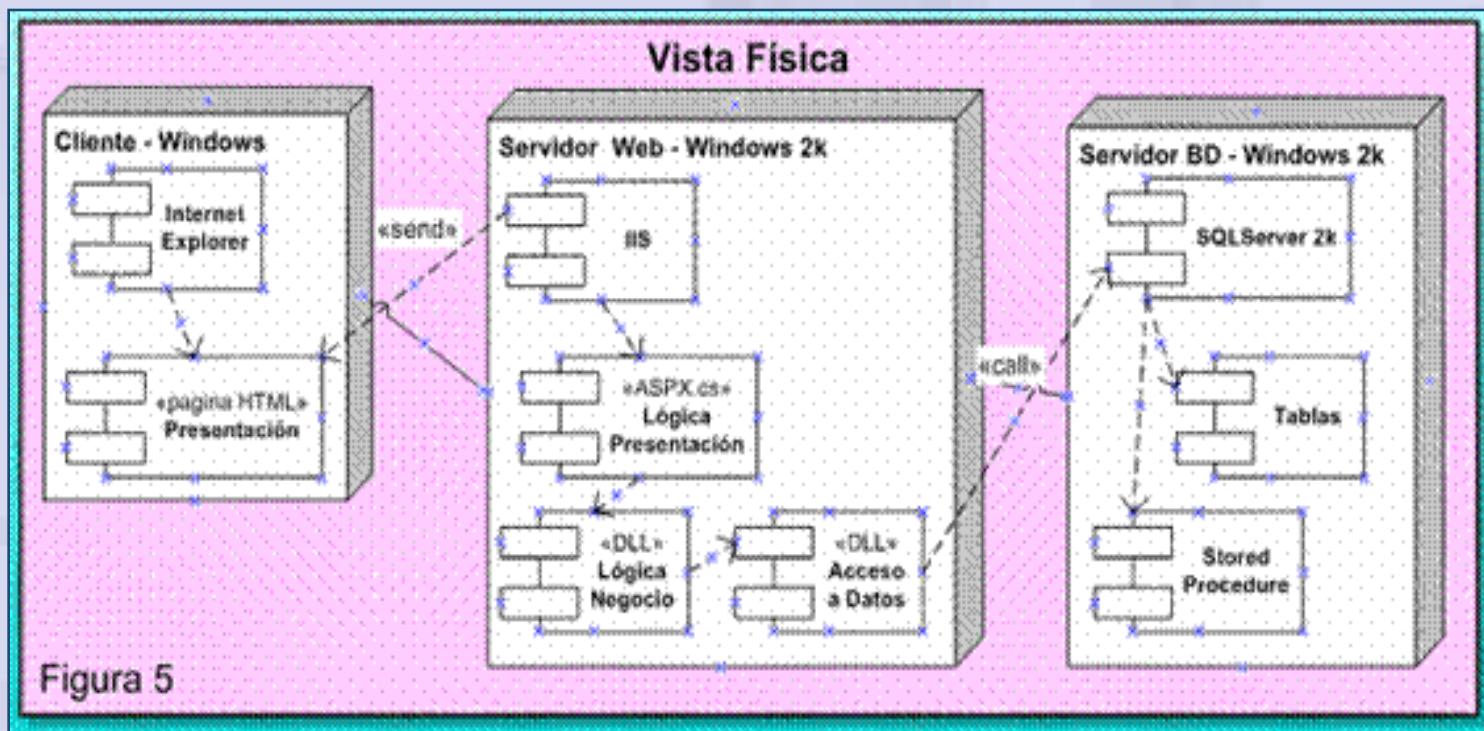
- En la actualidad, uno de los patrones de diseño más utilizado para cualquier tipo aplicaciones es el de Capas (Layers en inglés) donde, básicamente, se divide los elementos de diseño en paquetes de Interfaz de Usuario, Lógica de Negocio y Acceso a Datos y Servicios.
- Diagrama de Paquetes.



# Propuesta de Microsoft

## Vista Física

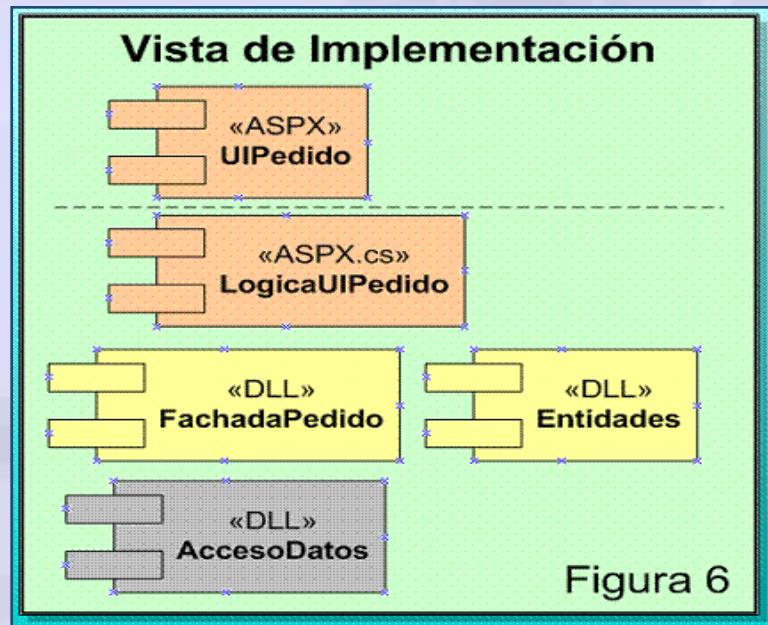
- Ilustra la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista lógica se "mapean" a componentes de software (servicios, procesos, etc.) o de hardware que definen más precisamente como se ejecutará la solución.
- Diagrama de Despliegue.



# Propuesta de Microsoft

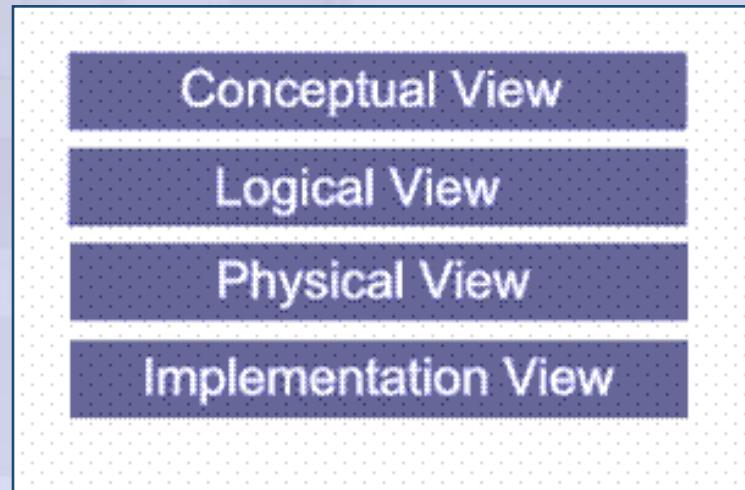
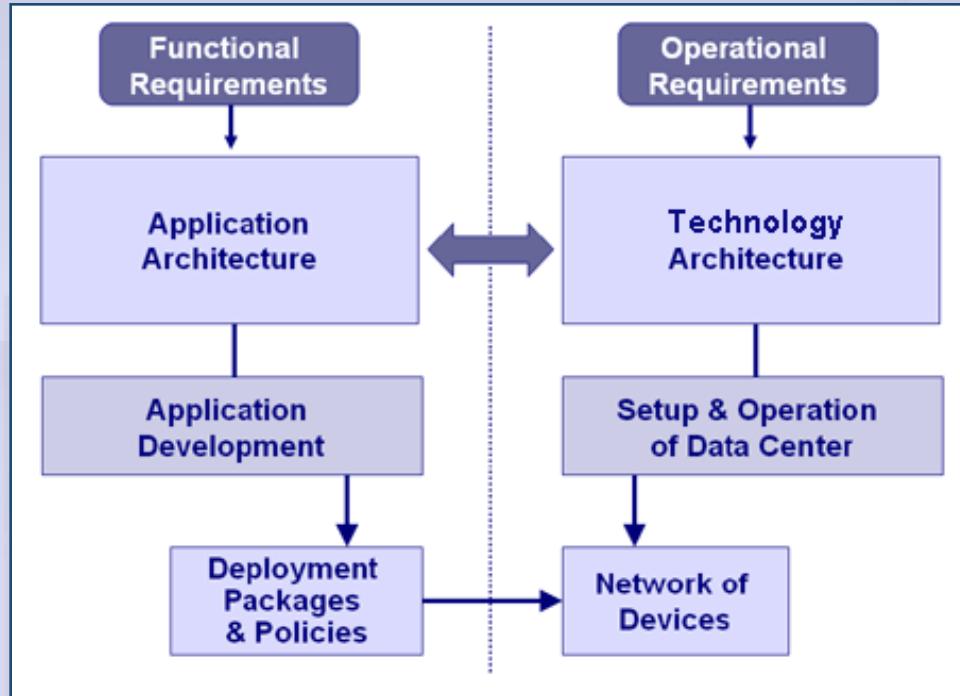
## Vista Implementación

- Describe cómo se implementan los componentes físicos mostrados en vista de distribución agrupándolos en subsistemas organizados en capas y jerarquías, ilustra, además las dependencias entre éstos. Básicamente, se describe el mapeo desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.
- Diagrama de Componentes.



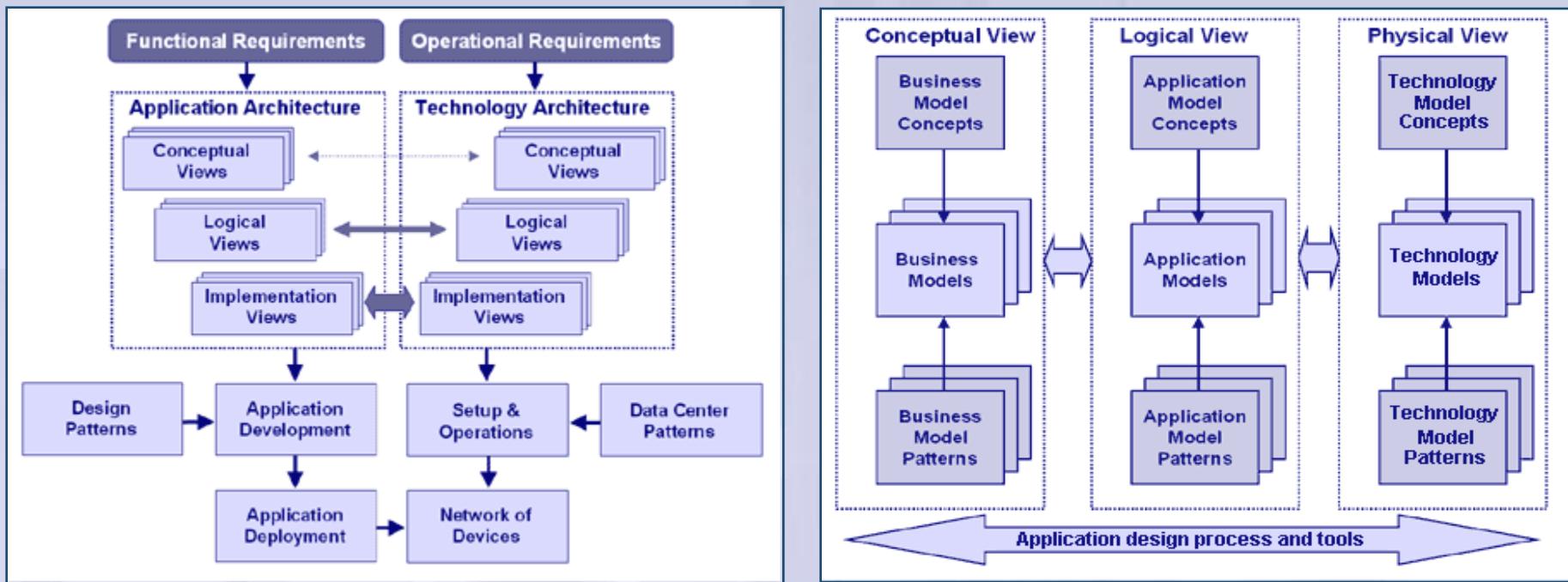
# Propuesta de Microsoft

## Implementación en la Arquitectura



# Propuesta de Microsoft

## Implementación en la Arquitectura



# Vistas y Diagramas de UML

| Área               | Vista                        | Diagramas                 | Conceptos principales                                                                |
|--------------------|------------------------------|---------------------------|--------------------------------------------------------------------------------------|
| Estructural        | Vista estática               | Diagrama de clases        | Clase, asociación, generalización, dependencia, realización, interfaz                |
|                    | Vista de casos de uso        | Diagramas de casos de uso | Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso |
|                    | Vista de implementación      | Diagrama de componentes   | Componente, interfaz, dependencia, realización                                       |
|                    | Vista de despliegue          | Diagrama de despliegue    | Nodo, componente, dependencia, localización                                          |
| Dinámica           | Vista de máquinas de estados | Diagrama de estados       | Estado, evento, transición, acción                                                   |
|                    | Vista de actividad           | Diagrama de actividad     | Estado, actividad, transición de terminación, división, unión                        |
|                    | Vista de interacción         | Diagrama de secuencia     | Interacción, objeto, mensaje, activación                                             |
|                    |                              | Diagrama de colaboración  | Colaboración, interacción, rol de colaboración, mensaje                              |
| Gestión del modelo | Vista de gestión del modelo  | Diagrama de clases        | Paquete, subsistema, modelo                                                          |

# Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

# Conclusiones generales

- Importancia de la Arquitectura de Software
- Criticidad de las decisiones tempranas de arquitectura
- Alto nivel de abstracción
- Vinculada con requerimientos no funcionales
- Fuerte impulso en la academia y la industria
- Herramientas arquitectónicas aún en proceso de definición y desarrollo
- Metodologías arquitectónicas, en proceso de elaboración preliminar
- Resta elaborar: tácticas arquitectónicas, métodos basados en arquitectura, vínculo entre conceptos de arquitectura, DSLs, factorías, building blocks, ...

# Problemas pendientes en AS

- Falta de criterio unificado
- No hay un modelo de proceso de punta a punta
- Desarrollo en paralelo de conceptos antagónicos o no coordinados
  - Métodos ágiles
  - Metodologías de ciclo de vida
  - Patrones, estilos y tácticas
- Apropiación nominal de la AS por estrategias que no implementan principios arquitectónicos pero se benefician de su prestigio
- Poca masa crítica de herramientas y lenguajes de modelado arquitectónico (de alto nivel, con conectores de primera clase)

# Metodologías arquitectónicas

- Posicionamiento en ciclo de vida (AS en RUP)
- Atributos de calidad & escenarios
- [Primitivas de atributo]
- Architecture Based Design (ABD)
- Tácticas Arquitectónicas
- Comparación de alternativas (SACAM)
- Diseño basado en atributos (ADD)
- Ventajas y desventajas (ATAM)
- Elección de arquitectura
- Análisis de arquitectura (SAAM)
- Quality Attribute Workshop (QAW)
- Revisión activa de diseño parcial (ARID)
- Análisis de costo/beneficio (CBAM)
- Reformulación: UML & RUP

# Beneficios

## ■ Decisiones tempranas

### ■ Barry Boehm, 1995

- Si un proyecto no ha logrado una arquitectura del sistema, incluyendo su justificación, el proyecto no debe empezar el desarrollo en gran escala. Si se especifica la arquitectura como un elemento a entregar, se la puede usar a lo largo de los procesos de desarrollo y mantenimiento [Boe95].

## ■ Análisis de consistencia antes de elaborar el diseño (y escribir el código)

## ■ Sistematización de la experiencia

## ■ Homogeneización del lenguaje (IEEE 1471)

## ■ Herramientas para la evolución

## ■ Re-utilización

# Agenda

1. Definiciones de Arquitectura de Software
2. Importancia de la Arquitectura de Software
3. Evolución de la Arquitectura de Software
4. El rol del Arquitecto de Software
5. Propuestas Arquitectónicas
6. Conclusiones
7. Referencias

# Referencias

- Carlos Billy Reynoso. Introducción a la Arquitectura de Software. Universidad de Buenos Aires <http://www.willydev.net/descargas/prev/IntroArq.pdf>
- Paul Reed. Reference Architecture: The best of best practices. IBM <http://www.ibm.com/developerworks/rational/library/2774.html>
- Amit Unde, Becoming an Architect in a System Integrator. Microsoft Corporation. <http://msdn.microsoft.com/en-us/architecture/cc505970.aspx>
- Pattern-Oriented Software Architecture, A System of Patterns, Volume 1, Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Wiley & Sons, 1996, ISBN 0 471 95869
- Patterns-Oriented Software Architecture, Volume 2 : Concurrent and Networked Objects, Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, 2000)
- Vitalie Temnenco. TOGAF or not TOGAF: Extending Enterprise Architecture beyond RUP. IBM <http://www.ibm.com/developerworks/rational/library/jan07/temnenco/index.html>

# Referencias

- Adrián Lasso. Arquitectura de Software. Microsoft Corporation.  
<http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp>
- Michael Platt. Microsoft Architecture Overview. Microsoft Corporation.  
<http://msdn.microsoft.com/architecture/overview/default.aspx?pull=/library/en-us/dnea/html/eaarchover.asp>
- Carlos Reynoso y Nicolás Kicillof. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Universidad de Buenos Aires.  
[http://www.microsoft.com/spanish/msdn/arquitectura/roadmap\\_arq/lenguaje.asp](http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.asp)
- Desarrollo de Software Basado en Componentes (Luis F. Iribarne Martínez, U. de Almerias, Tesis Doctoral, Capítulo 1 )
- Interoperabilidad e Integración, Forum de Desarrolladores Corporativos, Madrid, Diciembre del 2002.