



# DECSAI

**Departamento de Ciencias de la Computación e I.A.**

Universidad de Granada



## Arquitecturas software

© Fernando Berzal, [berzal@acm.org](mailto:berzal@acm.org)

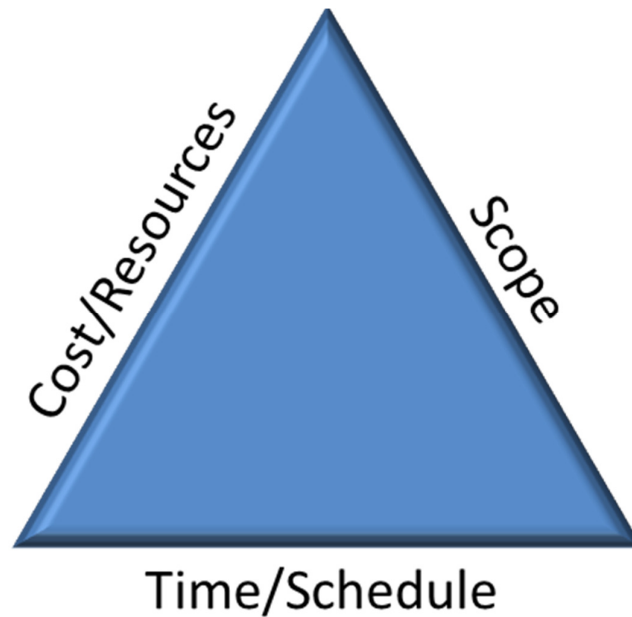
## Arquitecturas software



- Introducción: Conceptos clave.
- Vistas.
- Patrones arquitectónicos:
  - Arquitecturas basadas en capas.
  - Arquitecturas basadas en flujos de datos.
  - Arquitecturas con pizarra.
- Validación de la arquitectura.



# Introducción



"I can make it for you fast, cheap, or good. Pick any two."

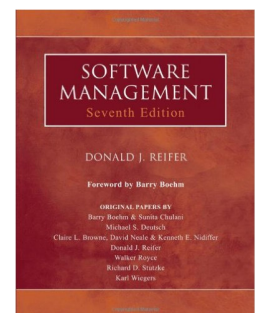
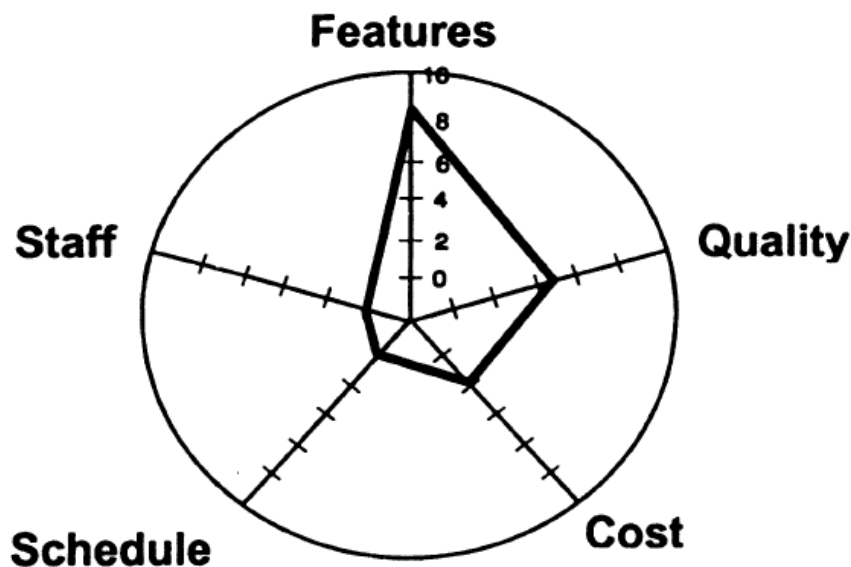
[https://en.wikipedia.org/wiki/Project\\_management\\_triangle](https://en.wikipedia.org/wiki/Project_management_triangle)



# Introducción



Representación mediante un diagrama de Kiviati:  
"Diagrama de flexibilidad"



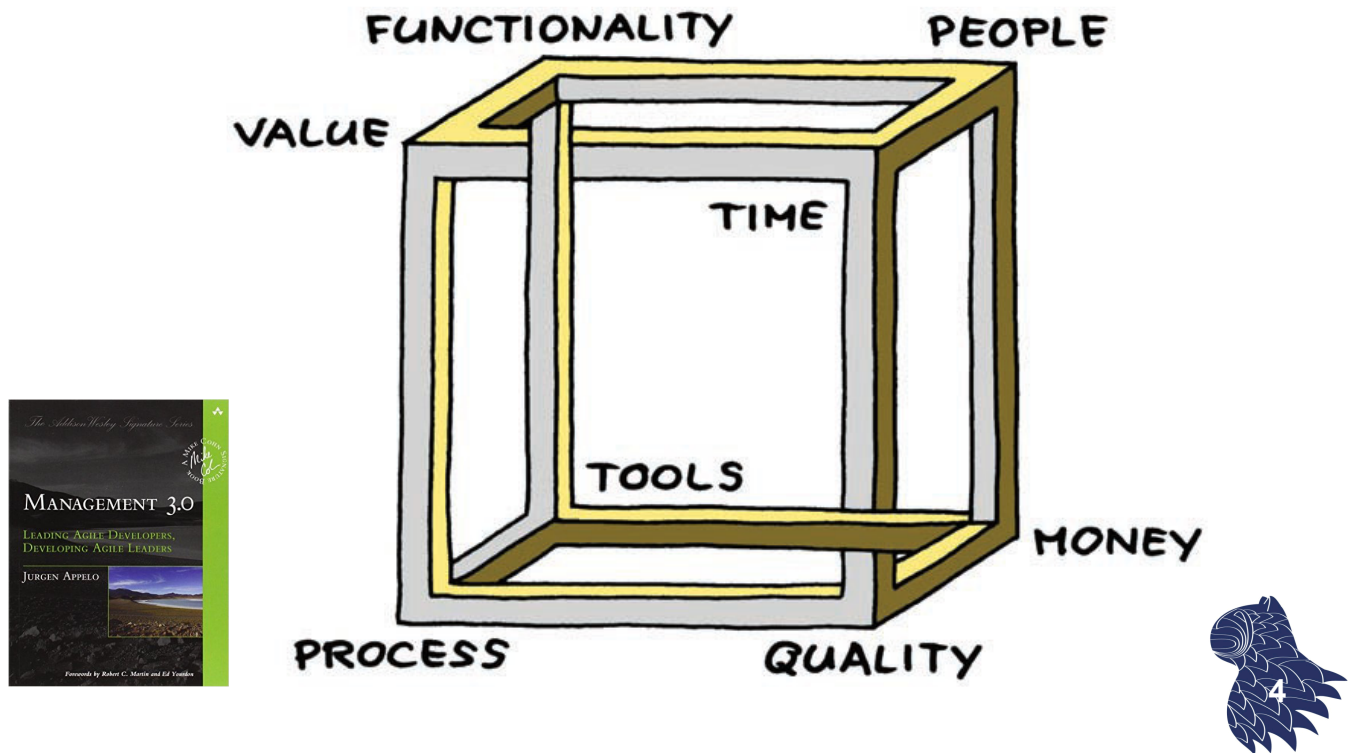
Karl E. Wieggers: "Creating a Software Engineering Culture", 1996



# Introducción



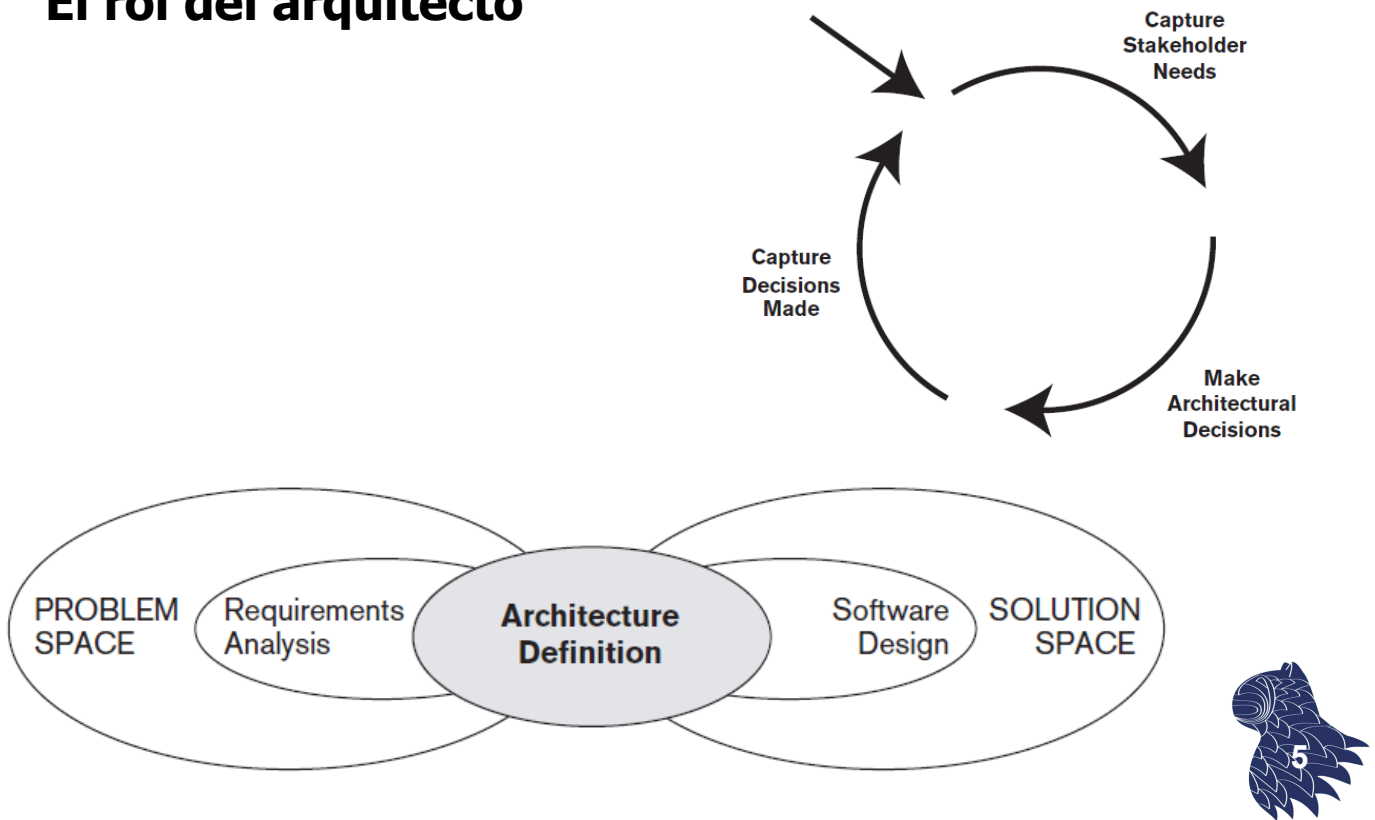
Puede ser difícil equilibrar todas las dimensiones clave...



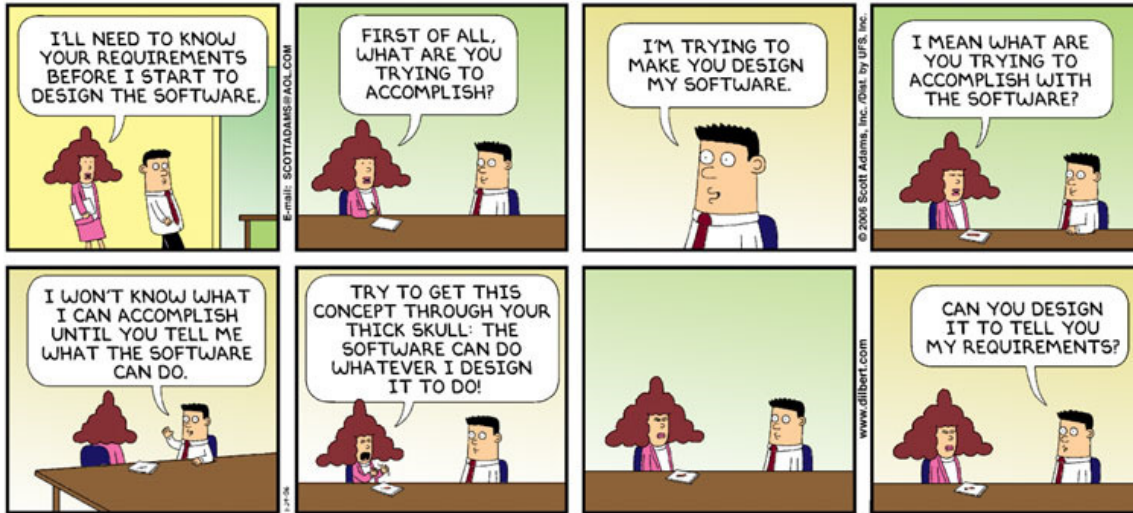
# Introducción



## El rol del arquitecto



# Introducción



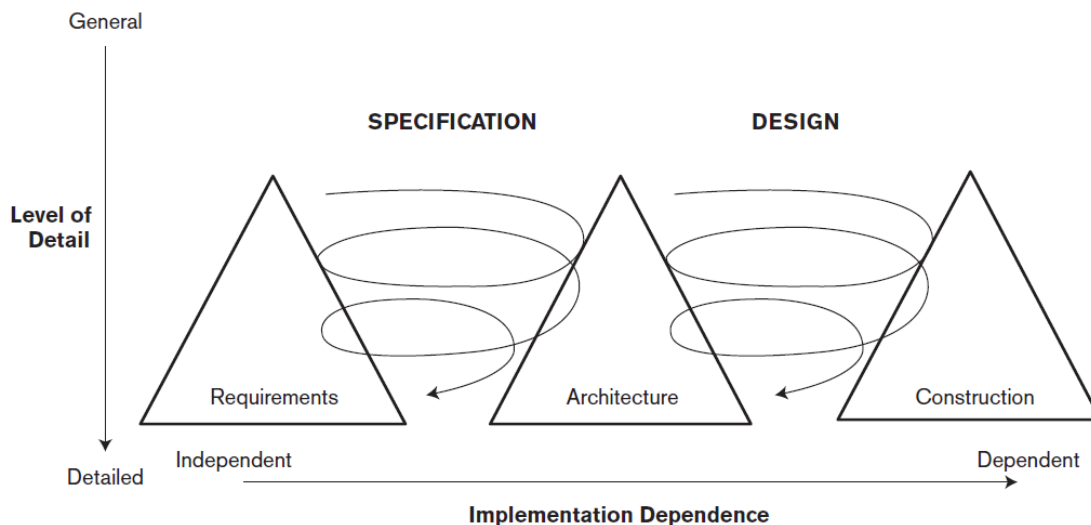
© Scott Adams, Inc./Dist. by UFS, Inc.



# Introducción



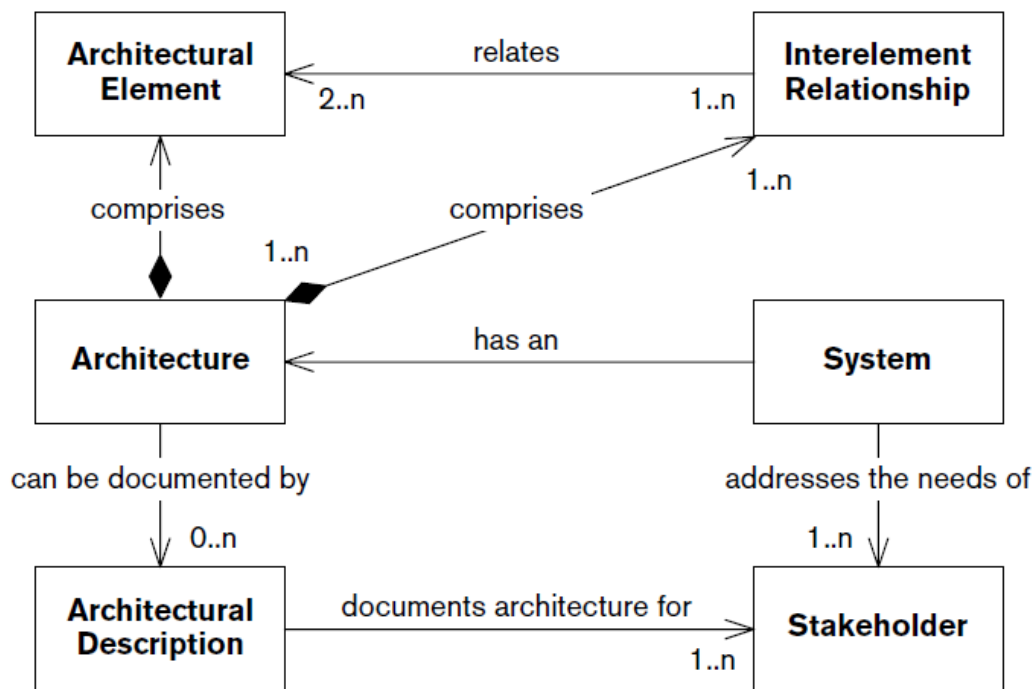
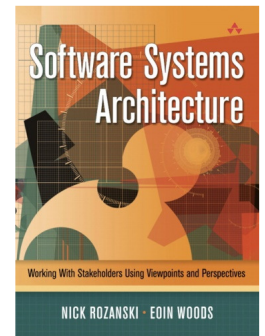
## El modelo de 3 picos



Bashar Nuseibeh:  
"Weaving Together Requirements and Architectures."  
IEEE Computer, 34(3):115–117, March 2001.



# Introducción



# Vistas



## OBSERVACIÓN

No es posible capturar todas las características funcionales y no funcionales de un sistema complejo en un único modelo completo que resulte comprensible.



## IDEA

Un sistema complejo se describe mucho mejor mediante un conjunto de vistas relacionadas entre sí, que de forma colectiva ilustran sus características y demuestran que satisface sus objetivos.

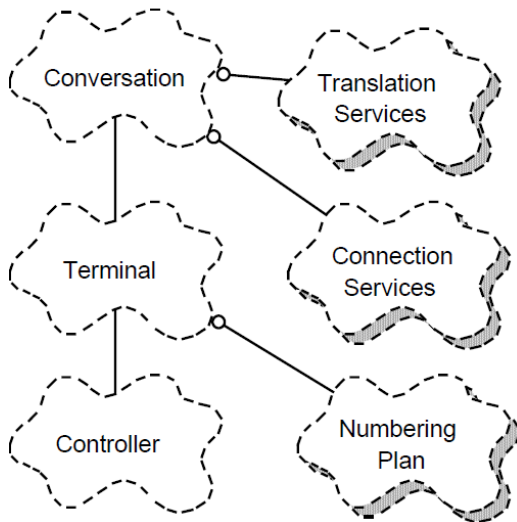




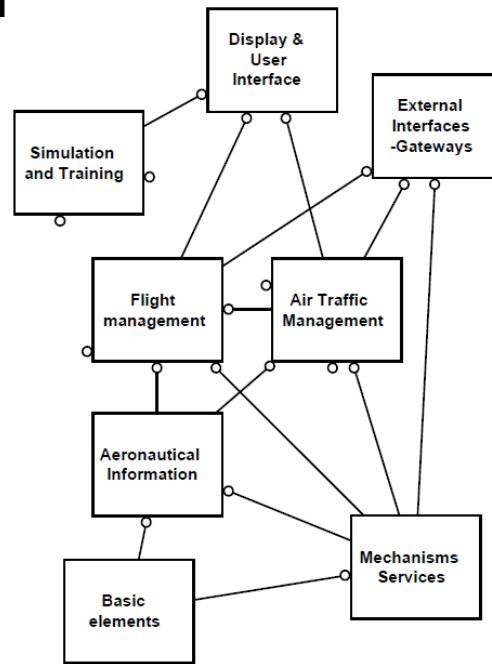


## Modelo 4+1: Vista lógica

Modelo OO del diseño



PABX

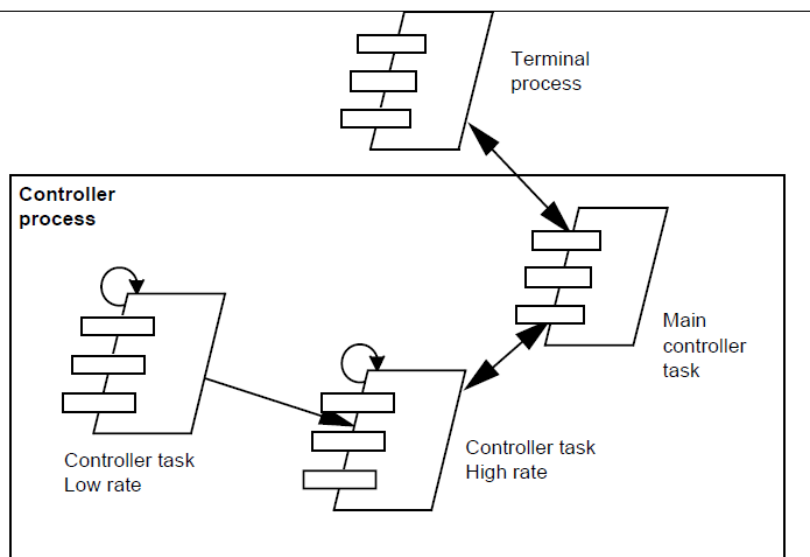


Control de tráfico aéreo



## Modelo 4+1: Vista de procesos

Concurrencia & sincronización



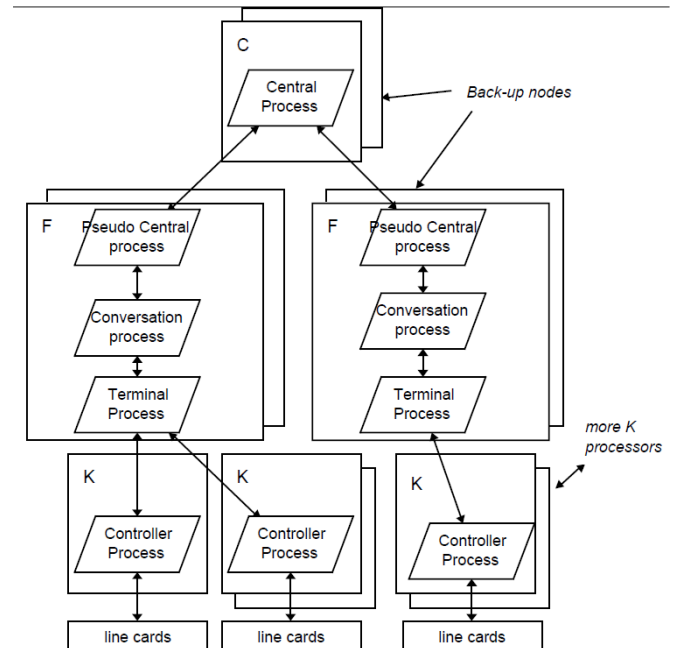
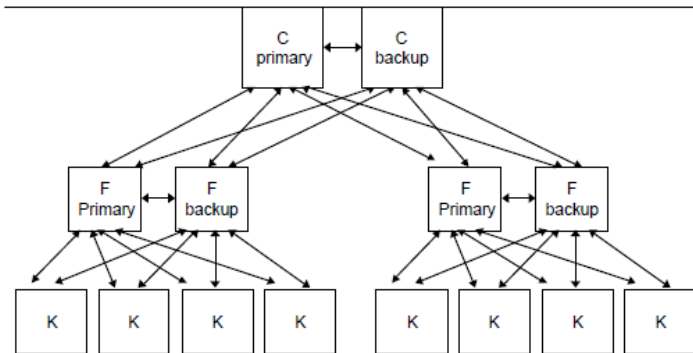
PABX





## Modelo 4+1: Vista física

Hardware & distribución

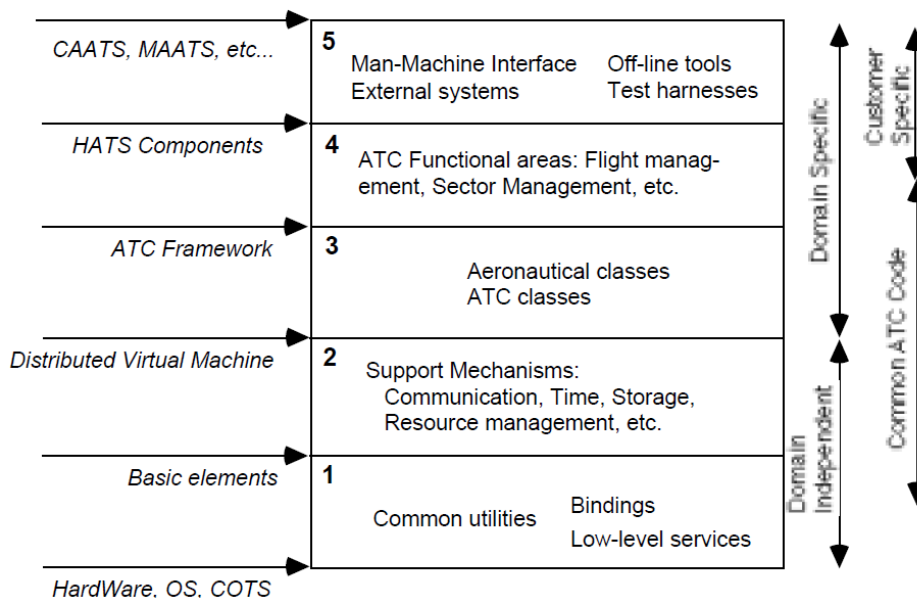


PABX



## Modelo 4+1: Vista de desarrollo

Organización de los módulos



Capas en un sistema de control de tráfico aéreo

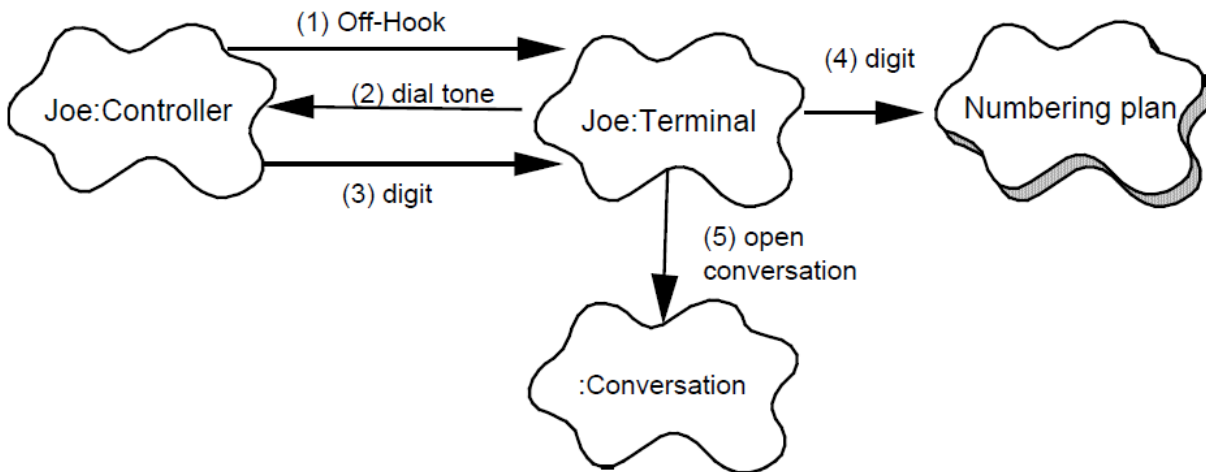






## Modelo 4+1: Escenarios

### Casos de uso seleccionados



PABX



## Modelo 4+1

### Resumen

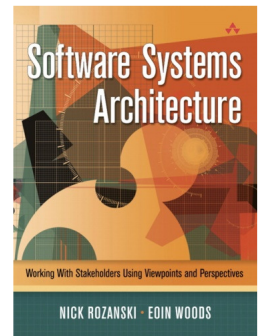
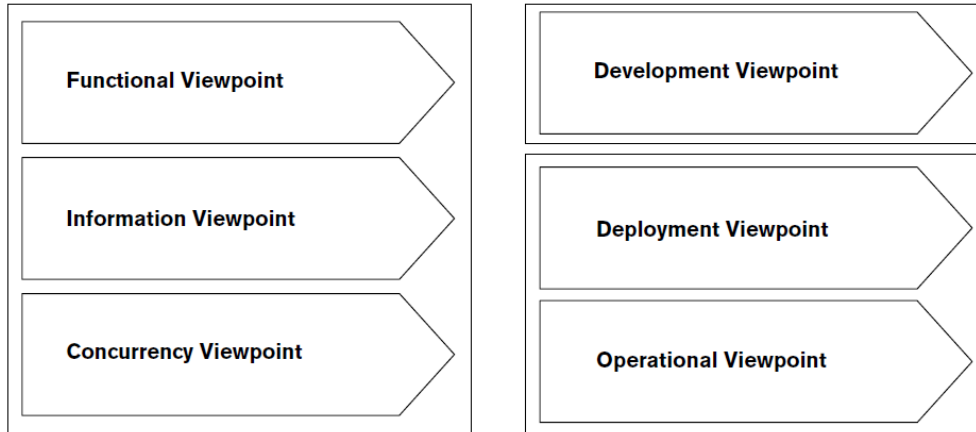
<i>View</i>	<i>Logical</i>	<i>Process</i>	<i>Development</i>	<i>Physical</i>	<i>Scenarios</i>
<i>Components</i>	Class	Task	Module, Subsystem	Node	Step, Scripts
<i>Connectors</i>	association, inheritance, containment	Rendez-vous, Message, broadcast, RPC, etc.	compilation dependency, "with" clause, "include"	Communication medium, LAN, WAN, bus, etc.	
<i>Containers</i>	Class category	Process	Subsystem (library)	Physical subsystem	Web
<i>Stakeholders</i>	End-user	System designer, integrator	Developer, manager	System designer	End-user, developer
<i>Concerns</i>	Functionality	Performance, availability, S/W fault-tolerance, integrity	Organization, reuse, portability, line-of-product	Scalability, performance, availability	Understandability
<i>Tool support</i>	Rose	UNAS/SALE DADS	Apex, SoDA	UNAS, Openview DADS	Rose





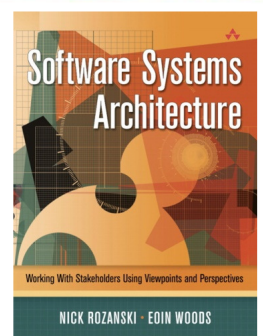
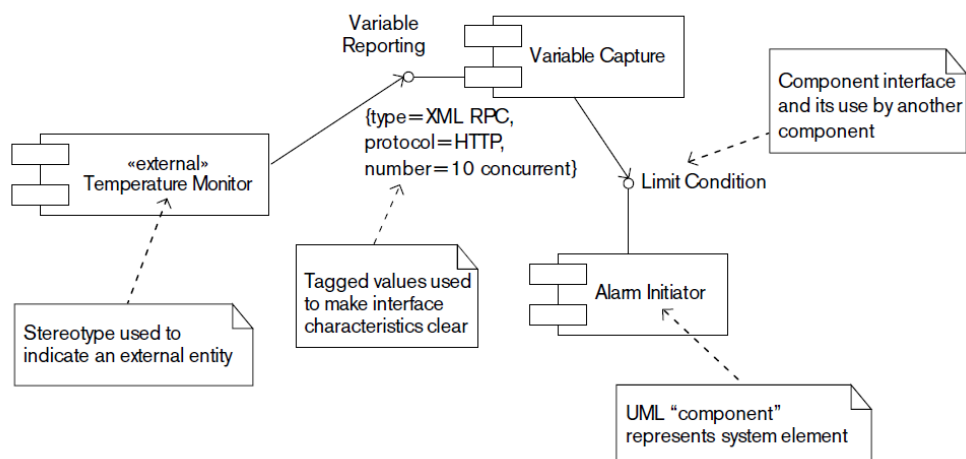
## Modelo alternativo

Puntos de vista...



## Modelo alternativo

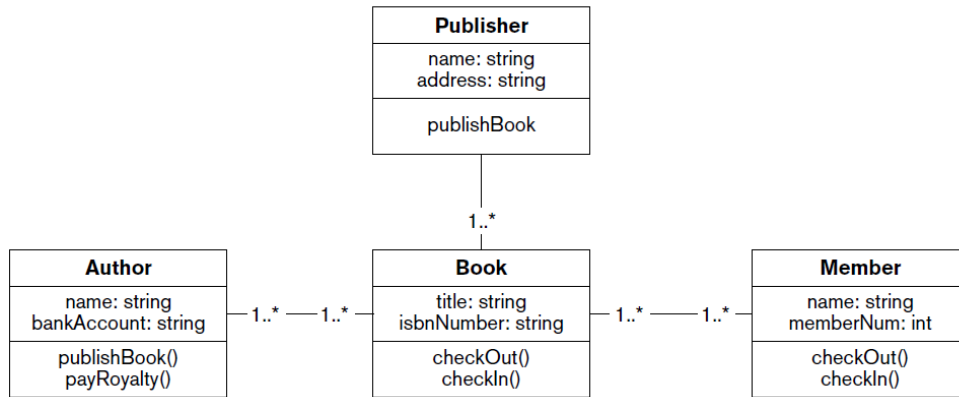
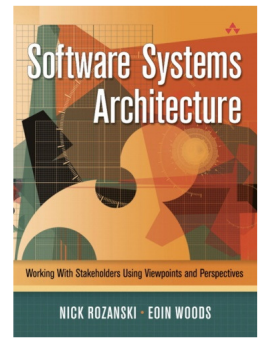
Punto de vista: Funcional





## Modelo alternativo

### Punto de vista: Información

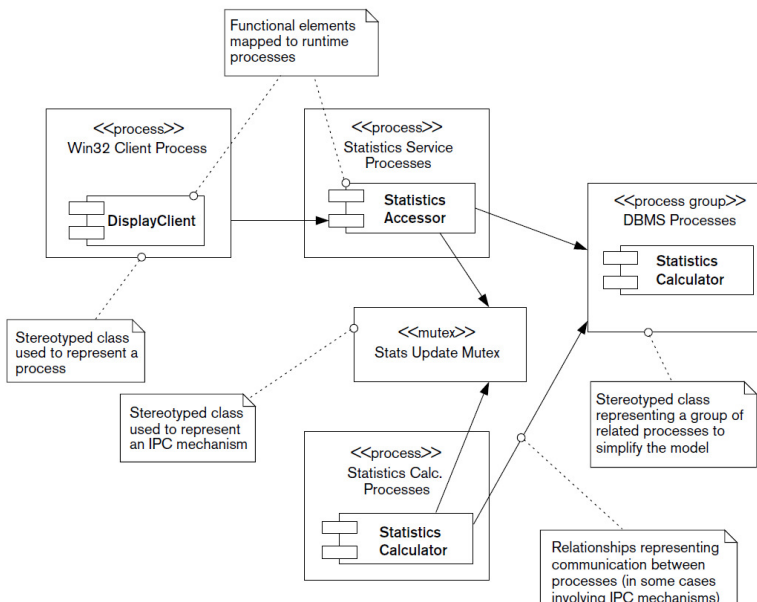
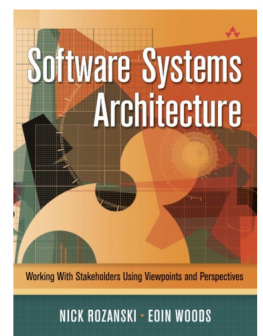


## Diagrama de clases UML



## Modelo alternativo

### Punto de vista: Concurrencia



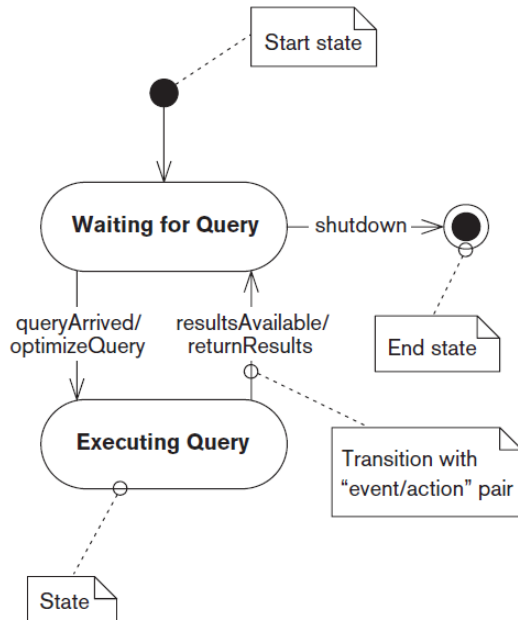
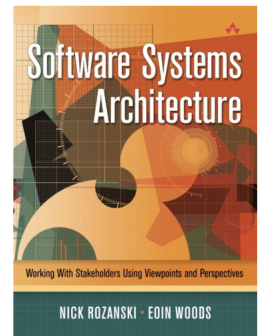
## Uso de estereotipos en UML





## Modelo alternativo

Punto de vista: Concurrencia

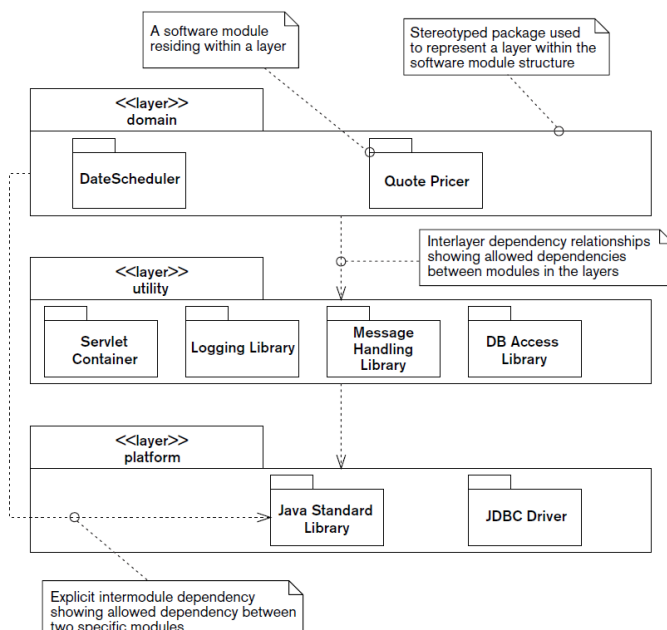
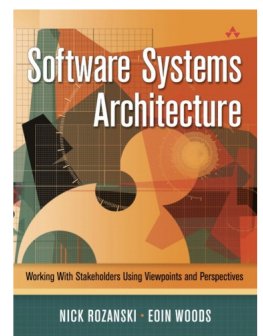


Diagramas de estados en UML



## Modelo alternativo

Punto de vista: Desarrollo

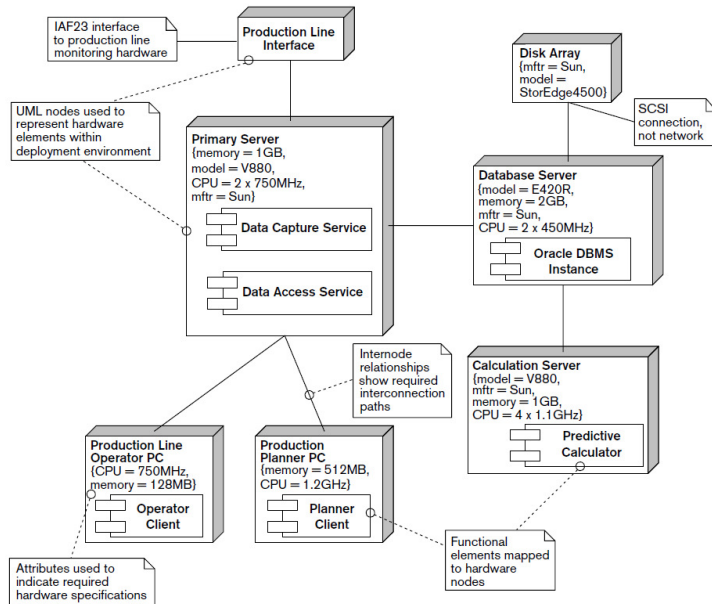
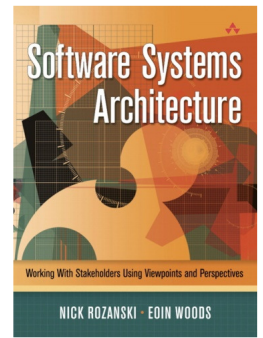


Diagramas de componentes en UML: Paquetes





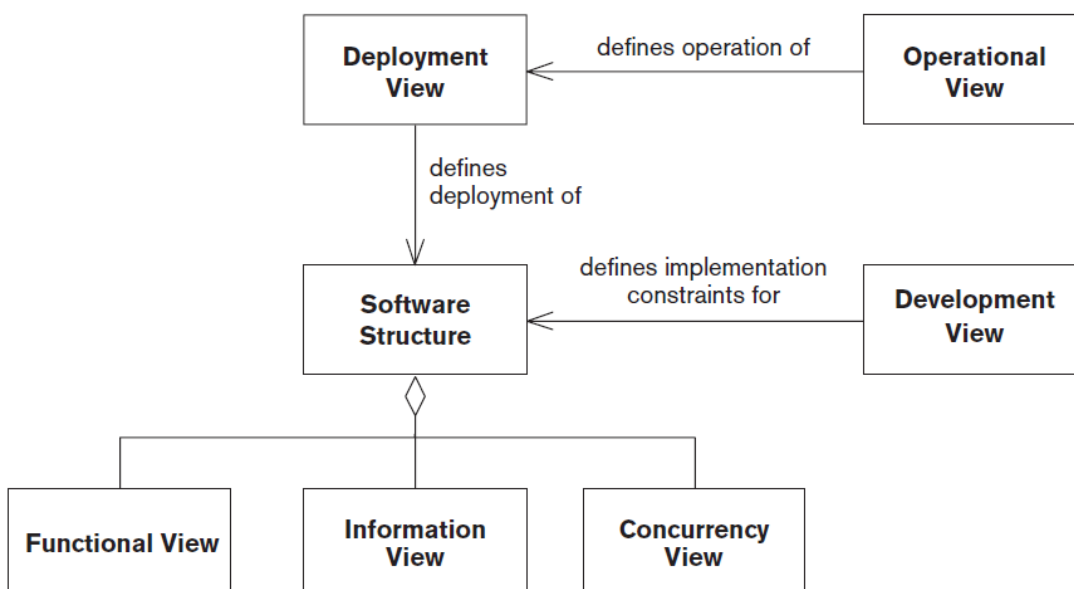
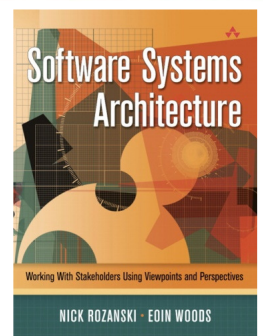
## Modelo alternativo Punto de vista: Despliegue



Diagramas de despliegue en UML



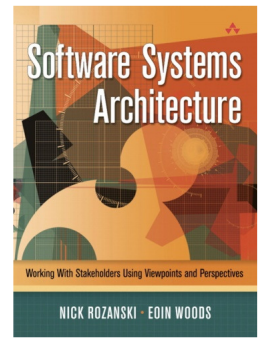
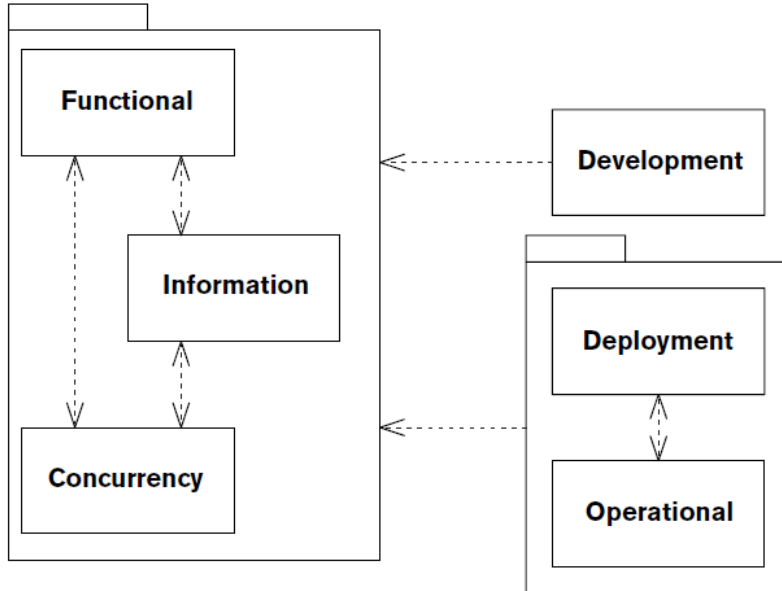
## Modelo alternativo Puntos de vista...





## Modelo alternativo

Puntos de vista...

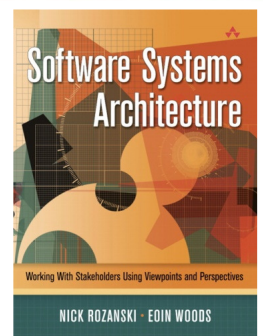
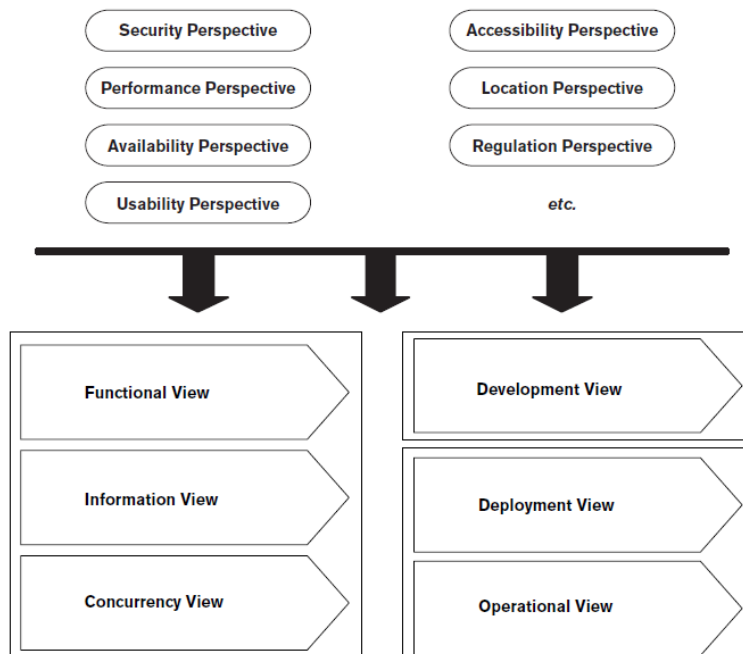


Dependencias entre puntos de vista



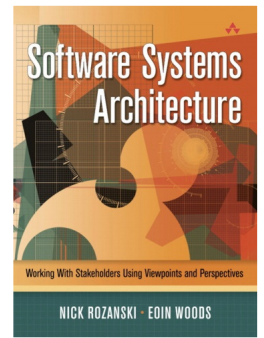
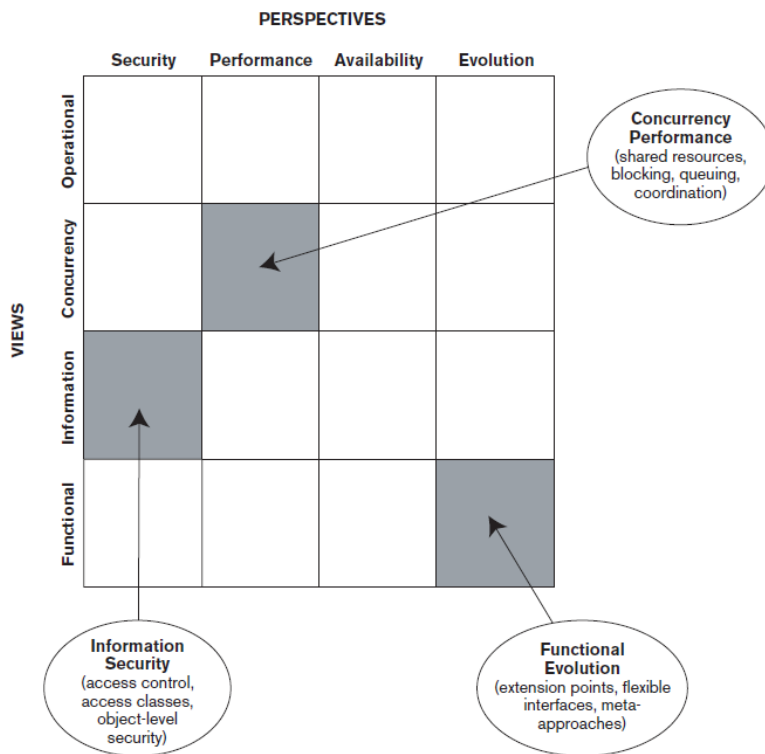
## Modelo alternativo

... y perspectivas

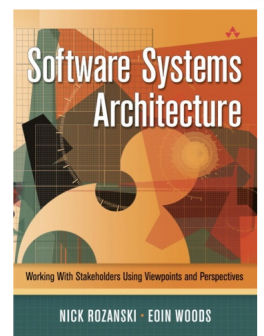
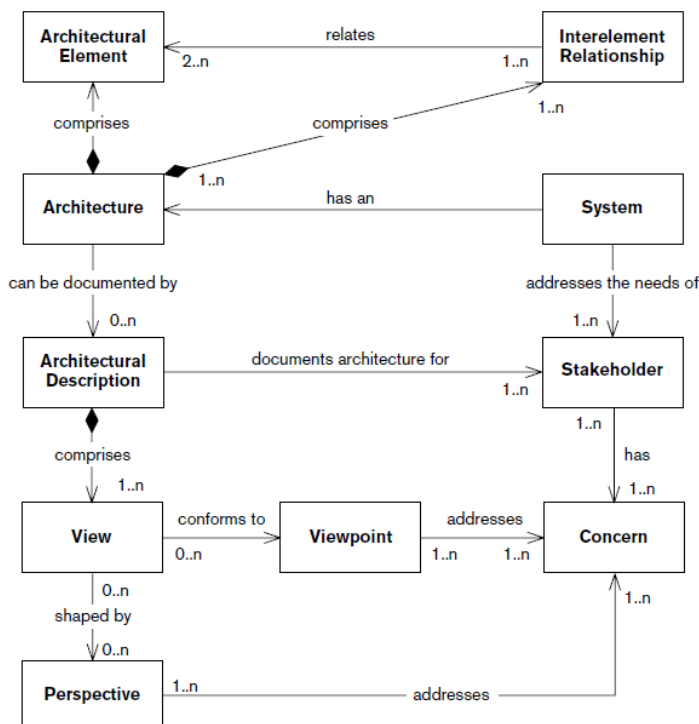




## Modelo alternativo



## Modelo alternativo





## Beneficios

- Separación de asuntos [separation of concerns].
- Comunicación con grupos de "stakeholders".
- Gestión de la complejidad del sistema.
- Descripción explícita de la arquitectura del sistema agrupando aspectos relevantes del sistema:  
Base del diseño del sistema.



## Limitaciones

- Inconsistencias entre las distintas vistas.
- Selección de un conjunto erróneo de vistas.
- Fragmentación de la descripción arquitectónica.





# Patrones arquitectónicos



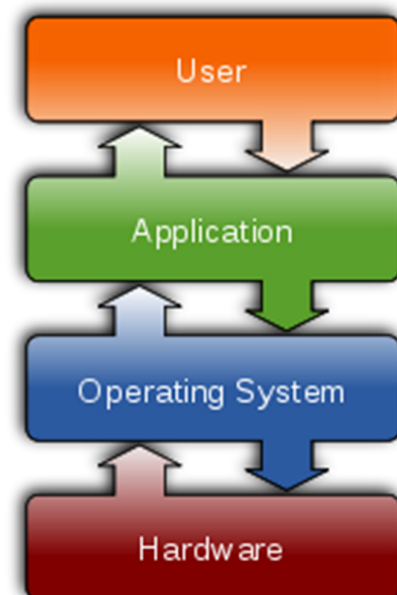
- Arquitecturas basadas en capas.
- Arquitecturas basadas en flujos de datos.
- Arquitecturas con pizarra



## Patrones arquitectónicos Capas



Ayudan a estructurar sistemas que pueden descomponerse en grupos de subtareas a distintos niveles de abstracción.

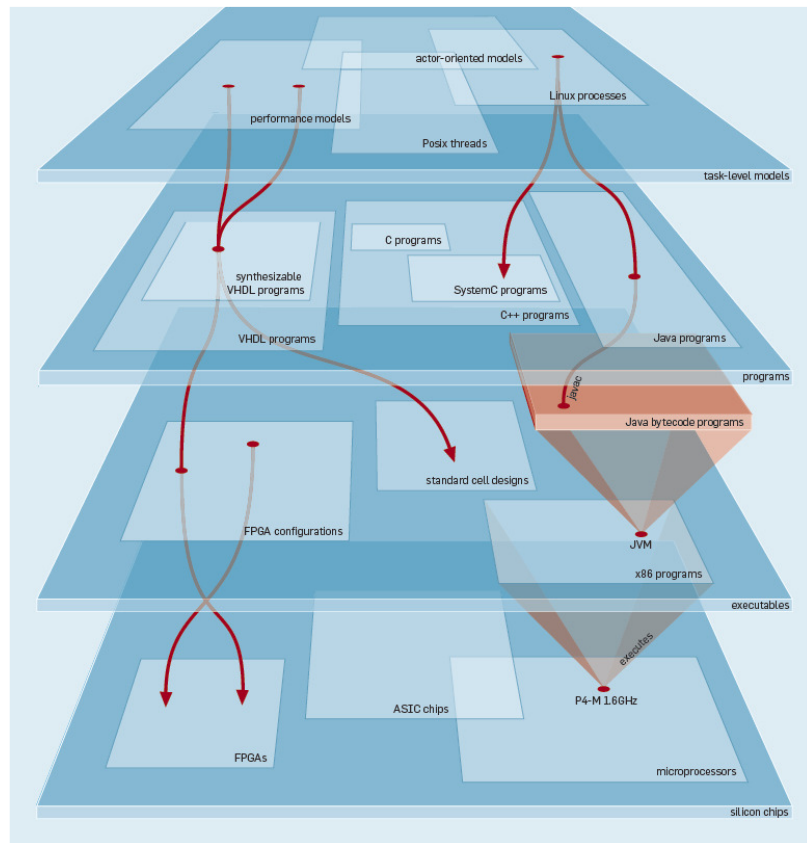


# Patrones arquitectónicos

## Capas



Capas de  
abstracción  
en computación



# Patrones arquitectónicos

## Capas



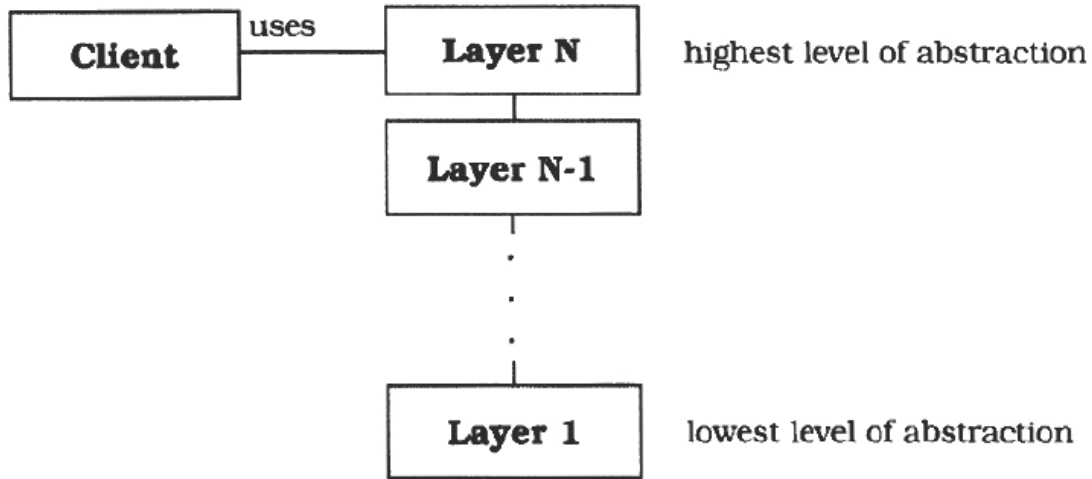
<p><b>Class</b> Layer J</p>	<p><b>Collaborator</b> • Layer J-1</p>
<p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Provides services used by Layer J+1.</li> <li>• Delegates subtasks to Layer J-1.</li> </ul>	

Tarjeta CRC de una capa [POSA]



# Patrones arquitectónicos

## Capas

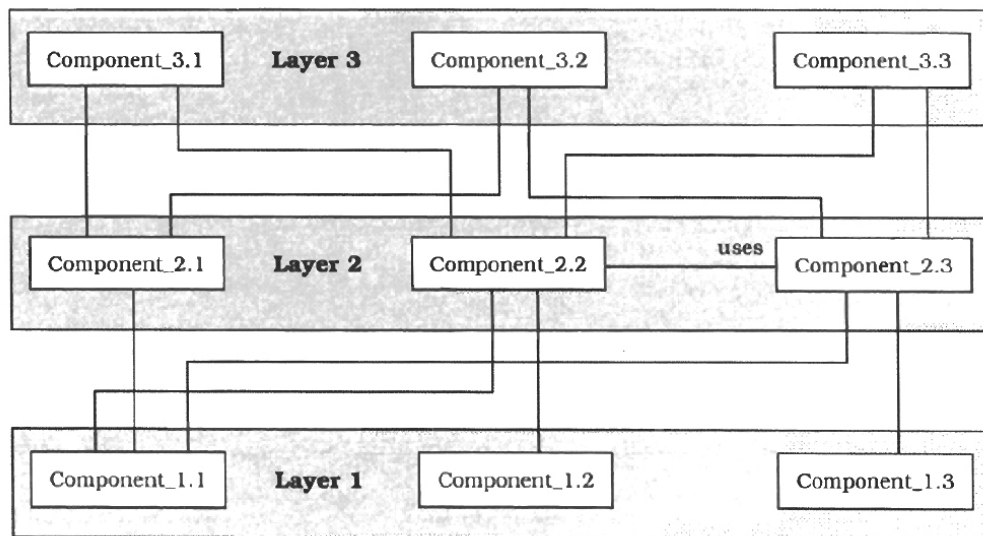


Estructura [POSA]



# Patrones arquitectónicos

## Capas

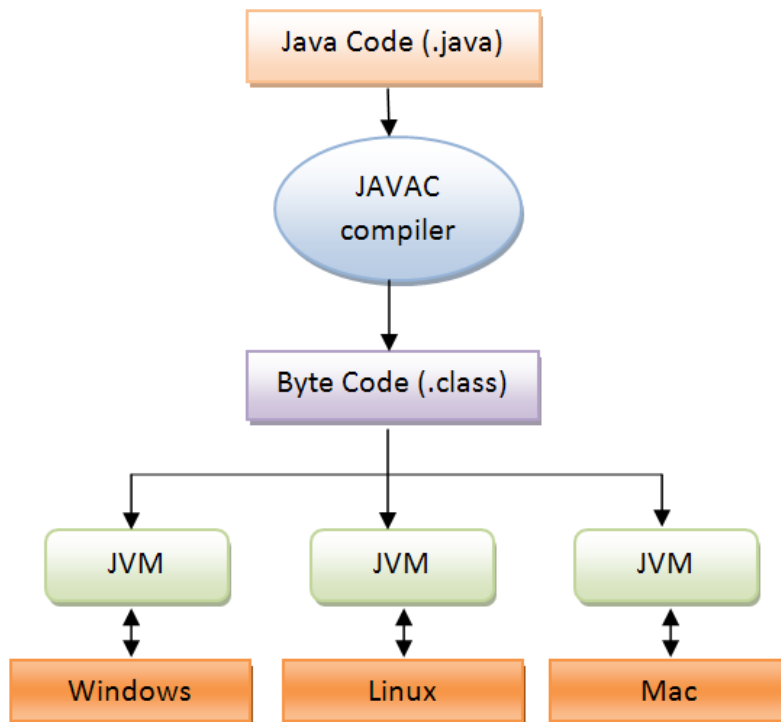


Estructura [POSA]

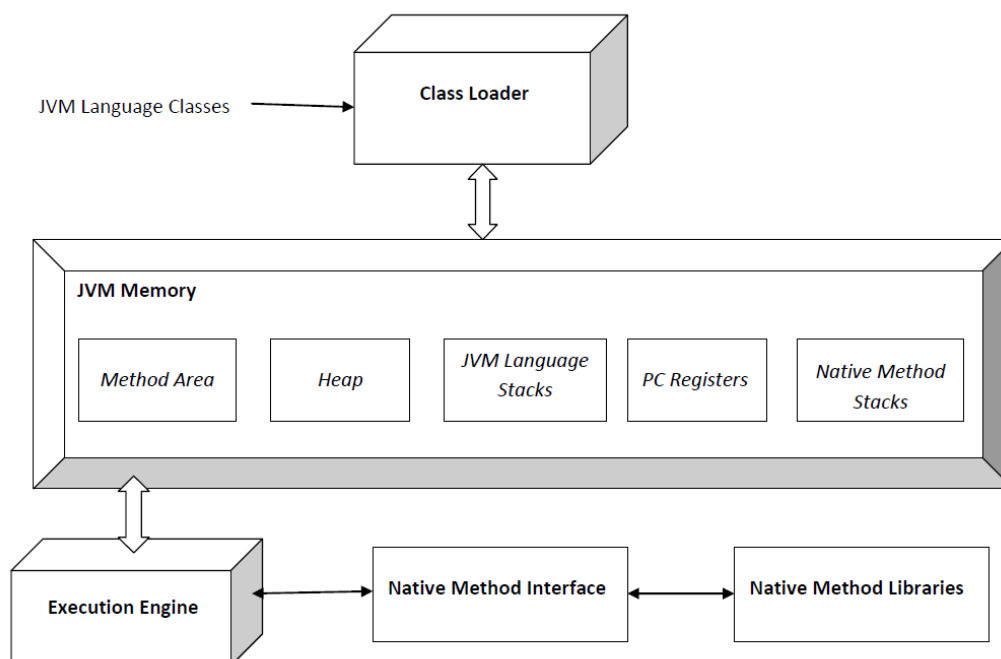




### La máquina virtual Java (JVM)



### La máquina virtual Java (JVM)

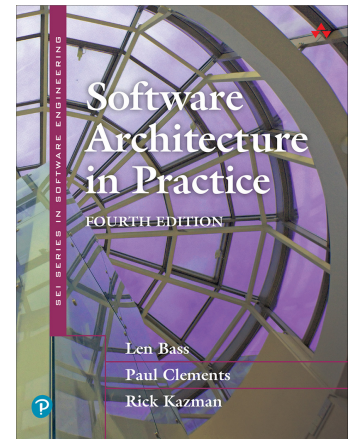
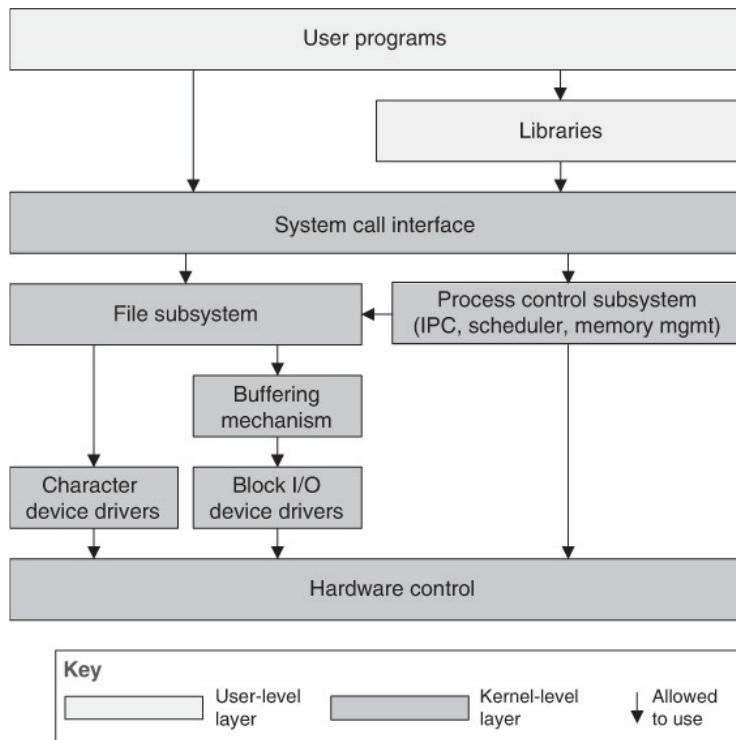


# Patrones arquitectónicos

## Capas



### Sistemas operativos



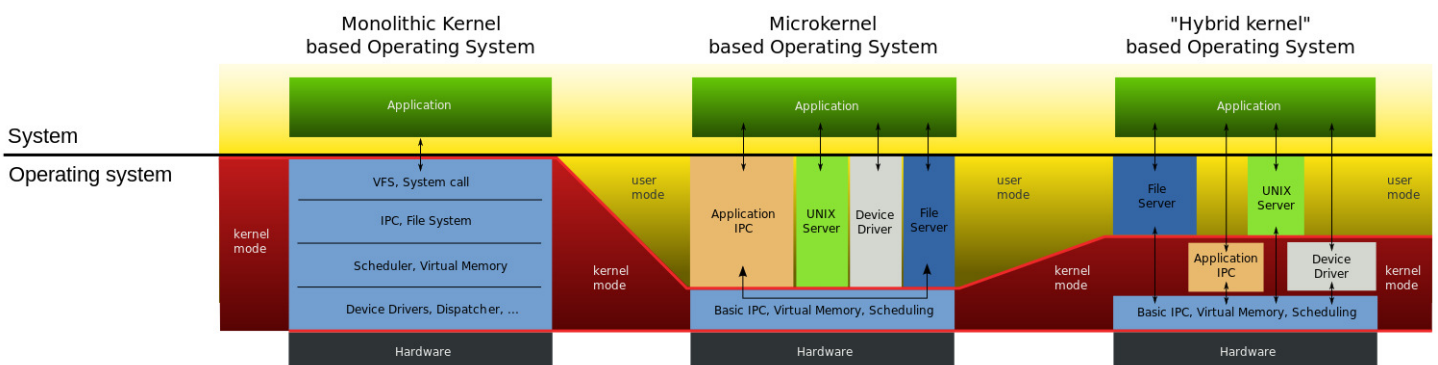
# Patrones arquitectónicos

## Capas



### Sistemas operativos

#### Kernel monolítico vs. Microkernel



Monolítico: BSD, AIX, Linux, Multics, Windows 9x...

Microkernel: Mach, RTOS (QNX, ChorusOS, Integrity)...

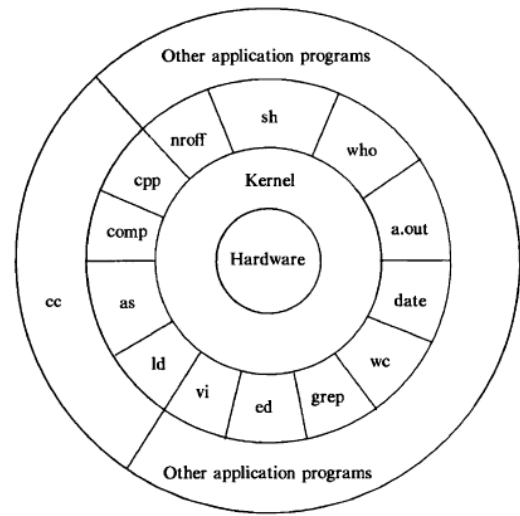
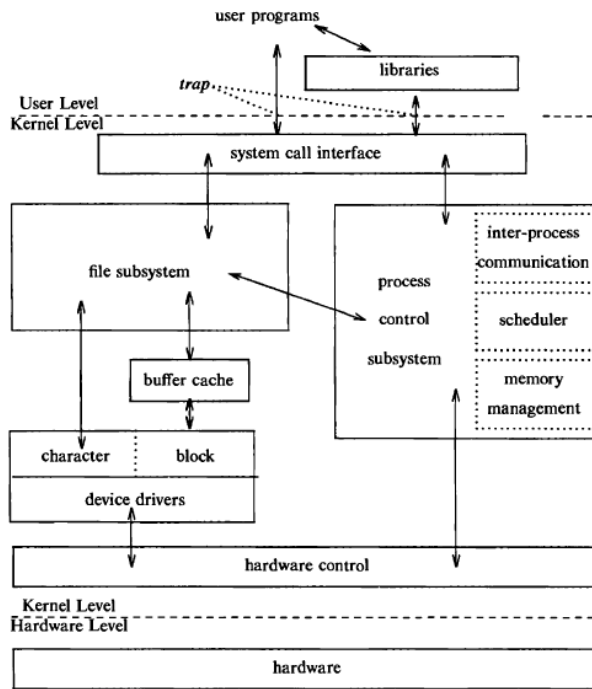
Híbrido: Windows NT...

[https://en.wikipedia.org/wiki/Kernel\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))





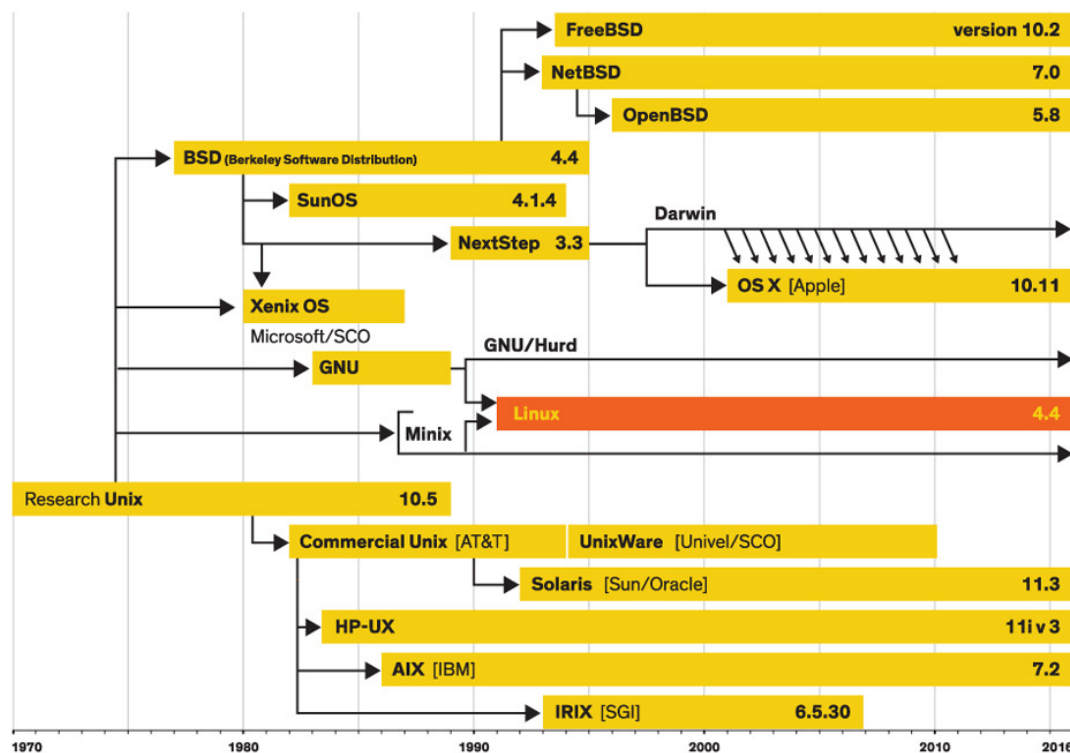
### Sistemas operativos: UNIX



Bach: The Design of the UNIX Operating System, 1986



### Sistemas operativos: UNIX

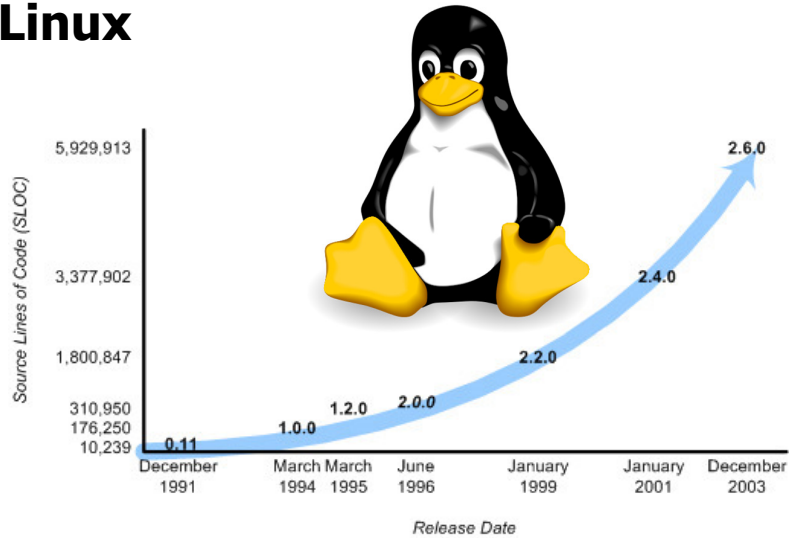
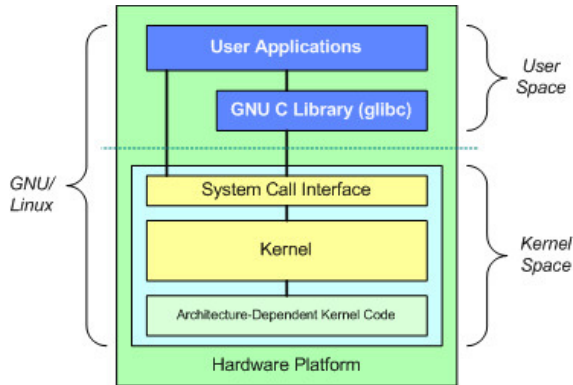


# Patrones arquitectónicos

## Capas



### Sistemas operativos: Linux



Versión 4.3 (2015)  
**>20MLOC**

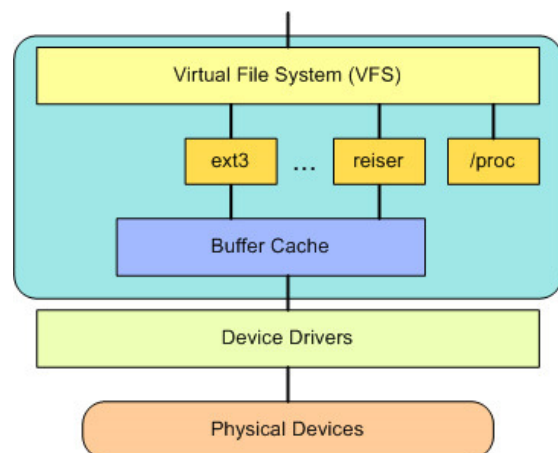
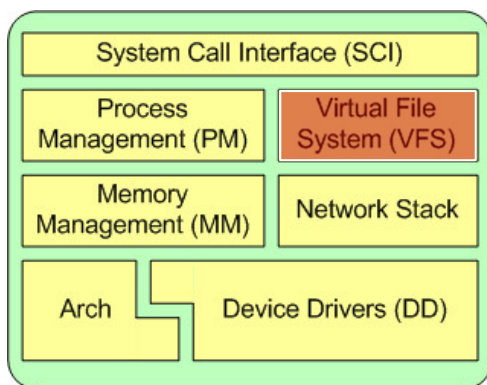


# Patrones arquitectónicos

## Capas



### Sistemas operativos: Linux

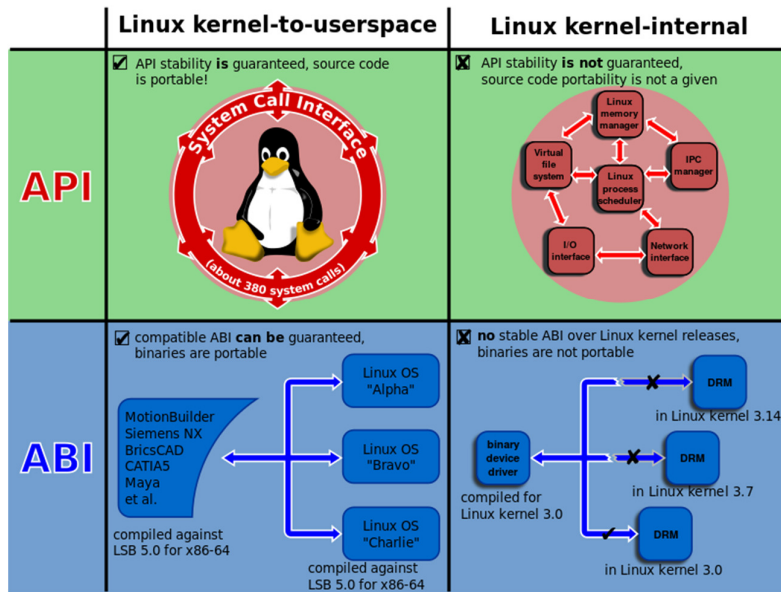




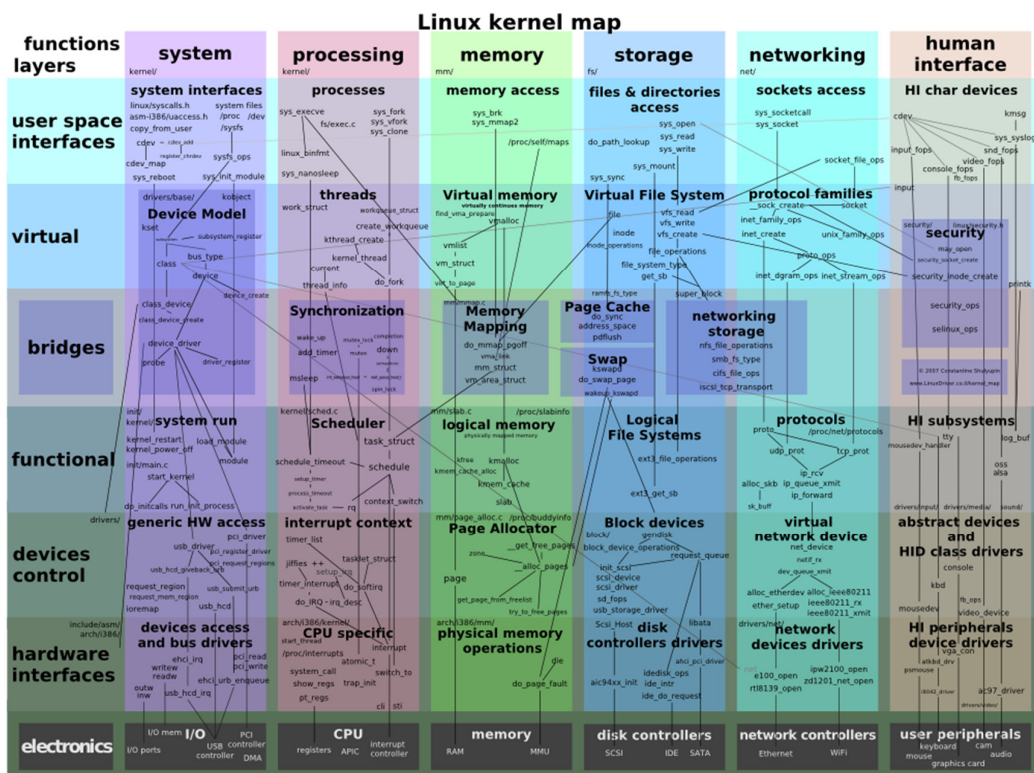
### Sistemas operativos: Linux

API: Application Programming Interface

ABI: Application Binary Interface



### Sistemas operativos: Linux





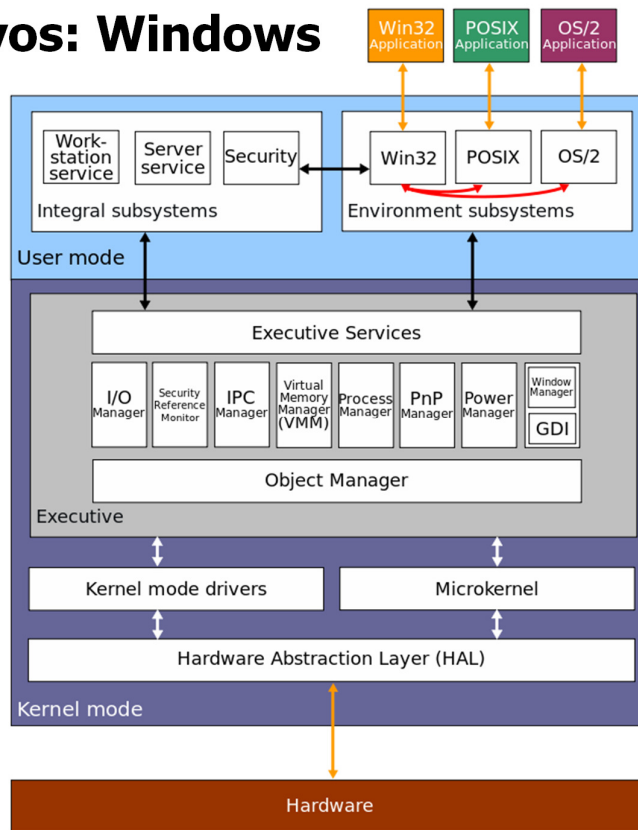
# Patrones arquitectónicos

## Capas



### Sistemas operativos: Windows

Windows NT  
(1989...)

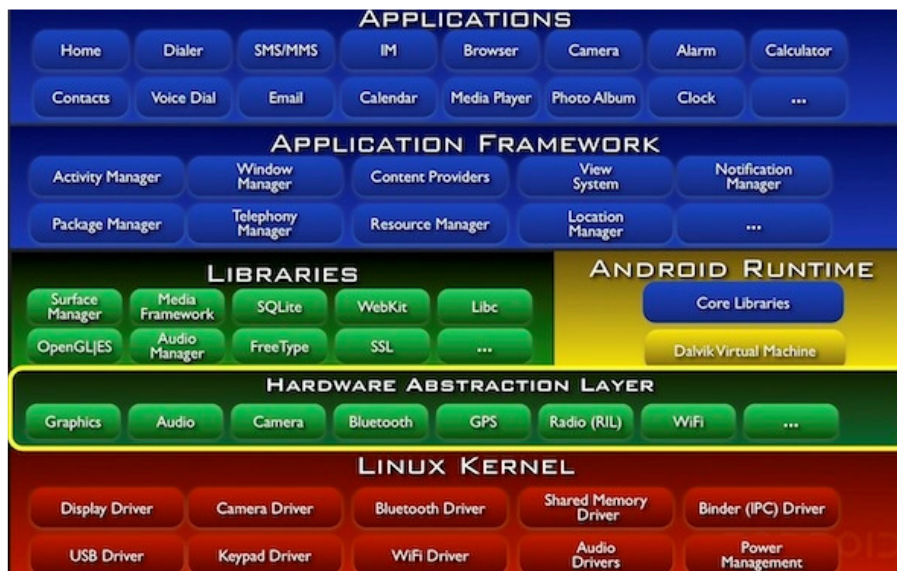


# Patrones arquitectónicos

## Capas

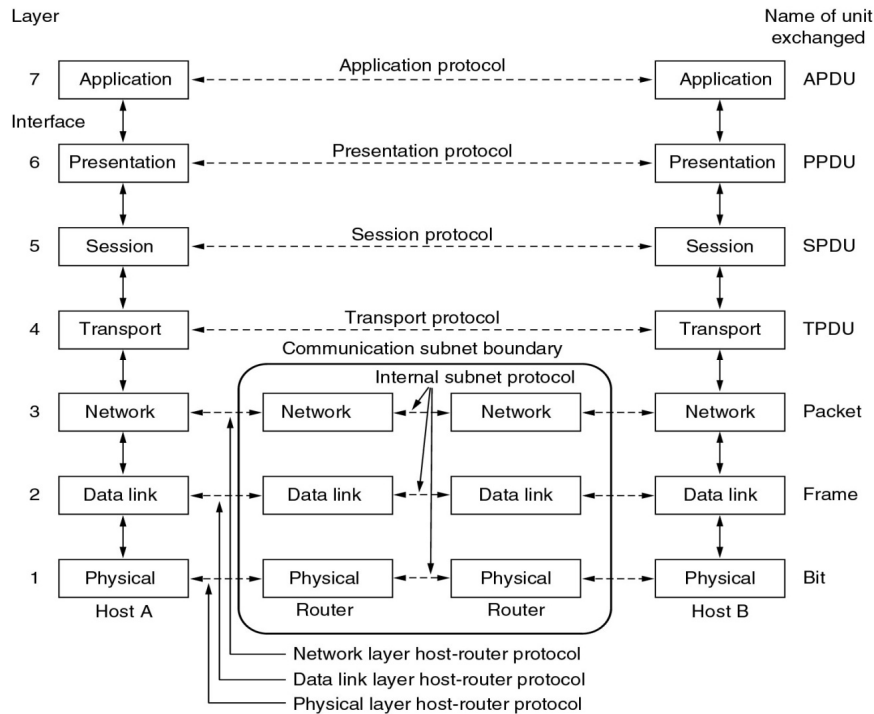


### Sistemas operativos: Android

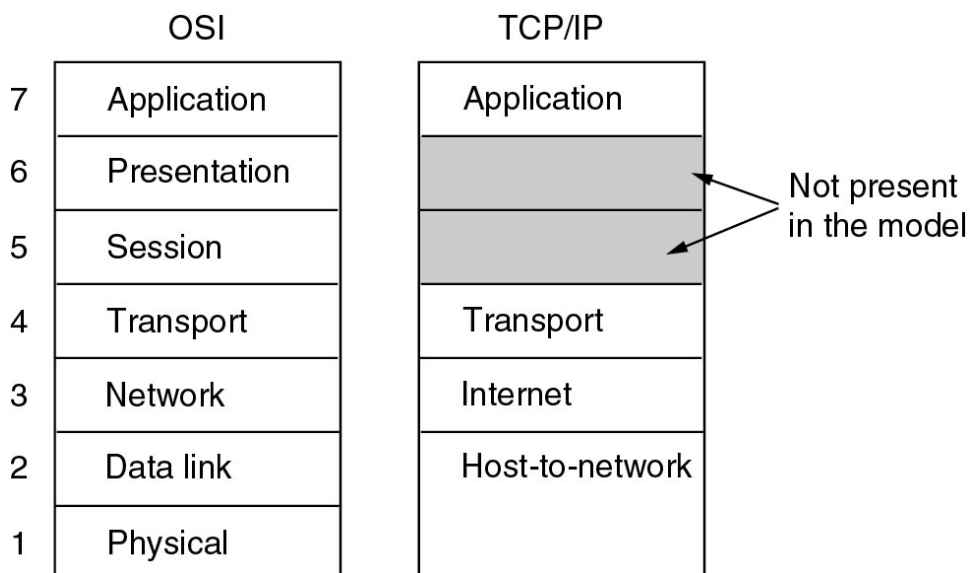




### Comunicaciones: El modelo de referencia OSI



### Comunicaciones: El modelo TCP/IP

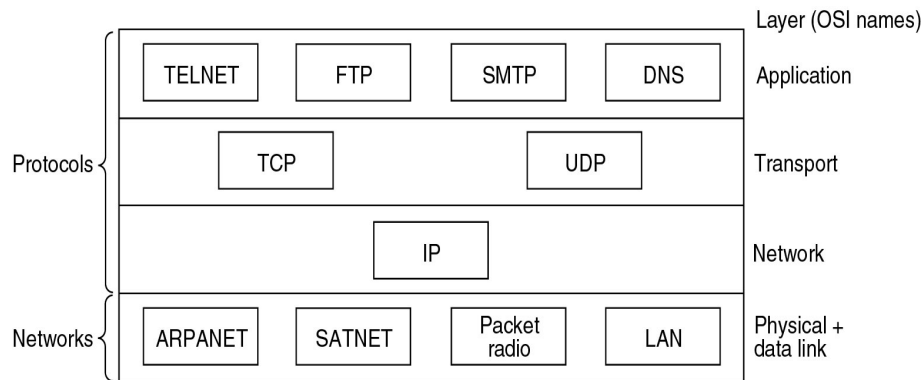


# Patrones arquitectónicos

## Capas



### Comunicaciones: El modelo TCP/IP

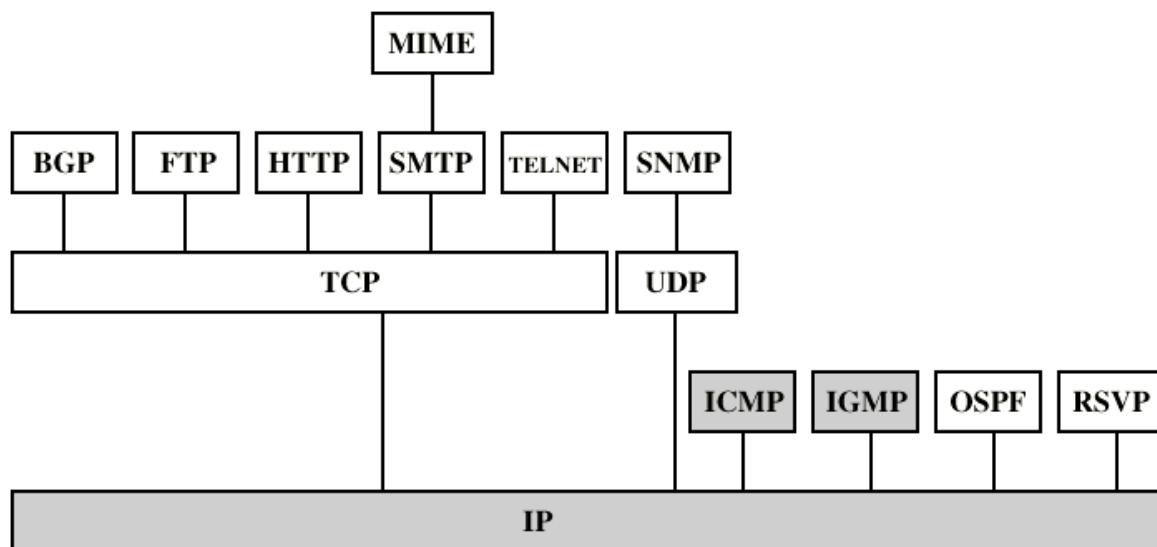


# Patrones arquitectónicos

## Capas



### Comunicaciones: El modelo TCP/IP



### Familia de protocolos TCP/IP

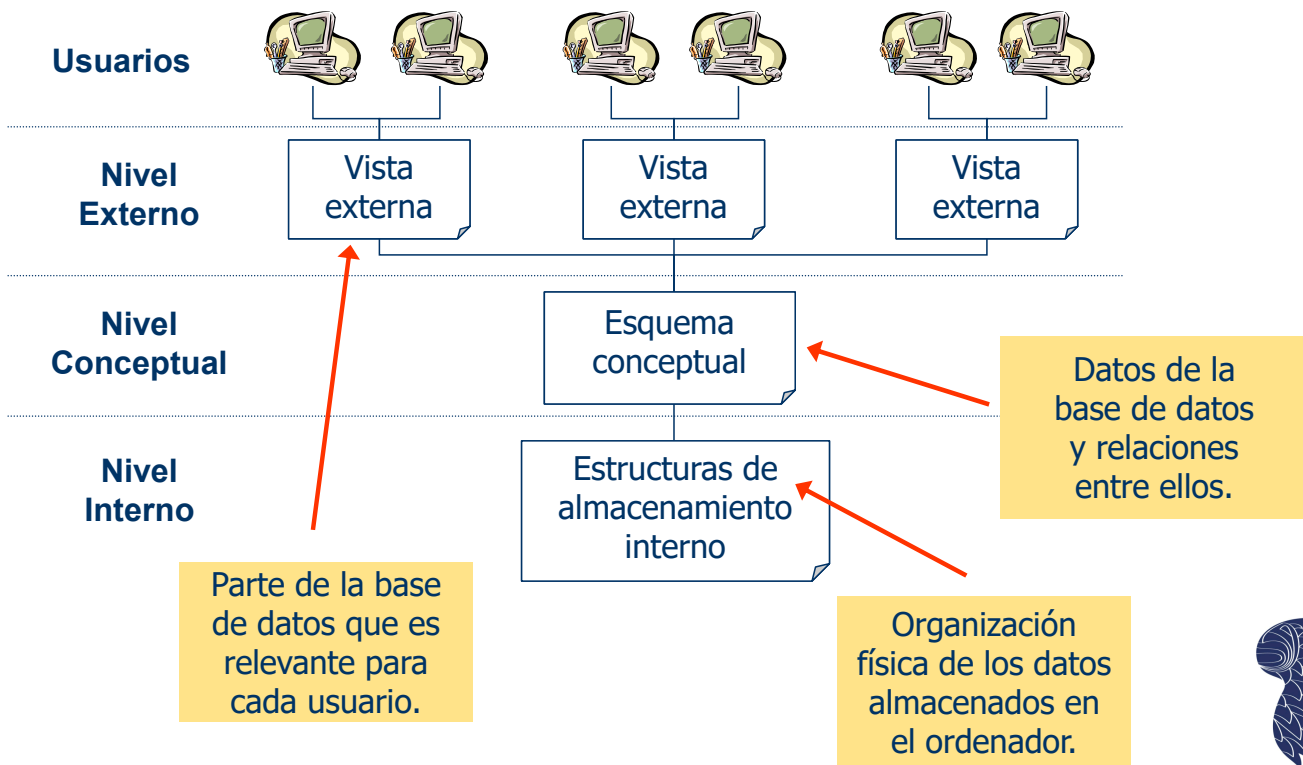


# Patrones arquitectónicos

## Capas



### Arquitectura ANSI/SPARC de un DBMS

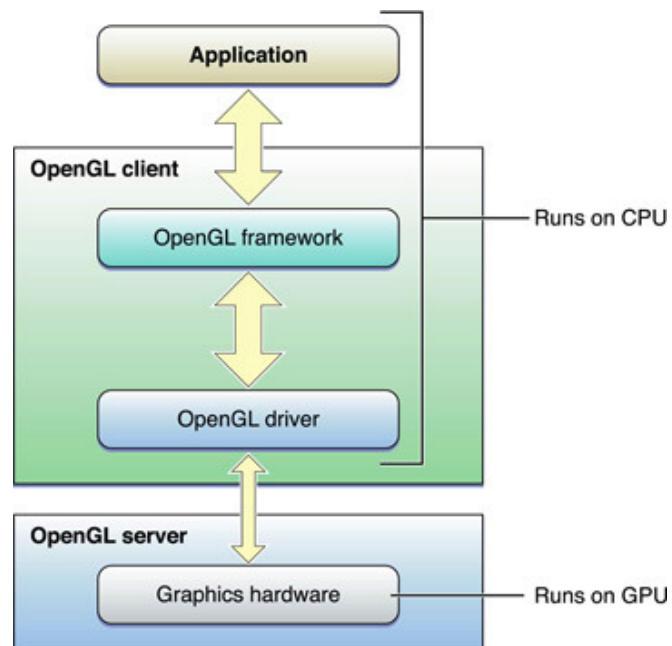
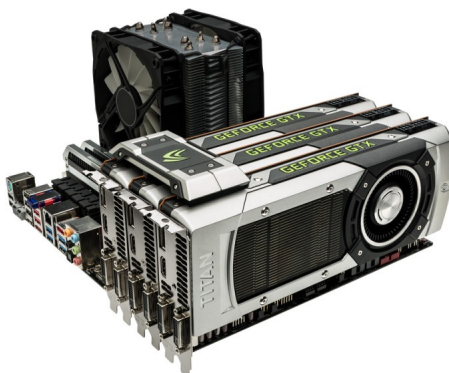
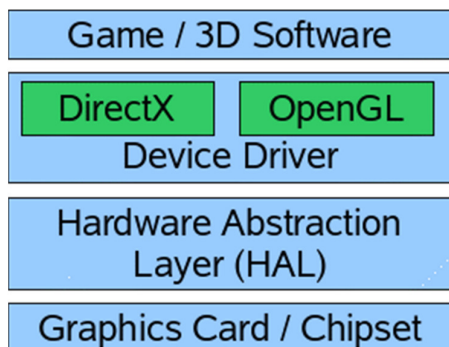


# Patrones arquitectónicos

## Capas



### OpenGL/DirectX

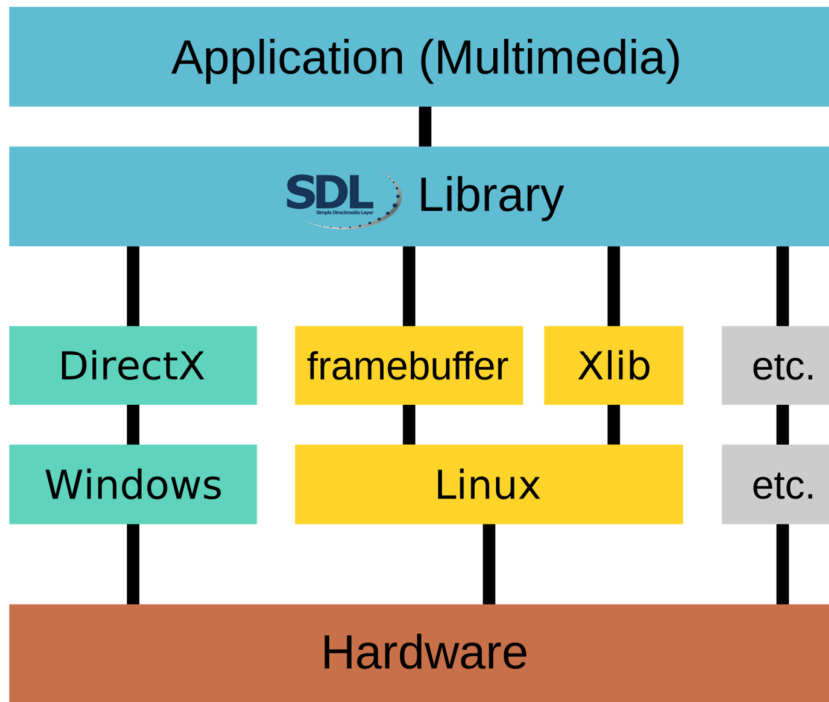


# Patrones arquitectónicos

## Capas



### Motores de videojuegos



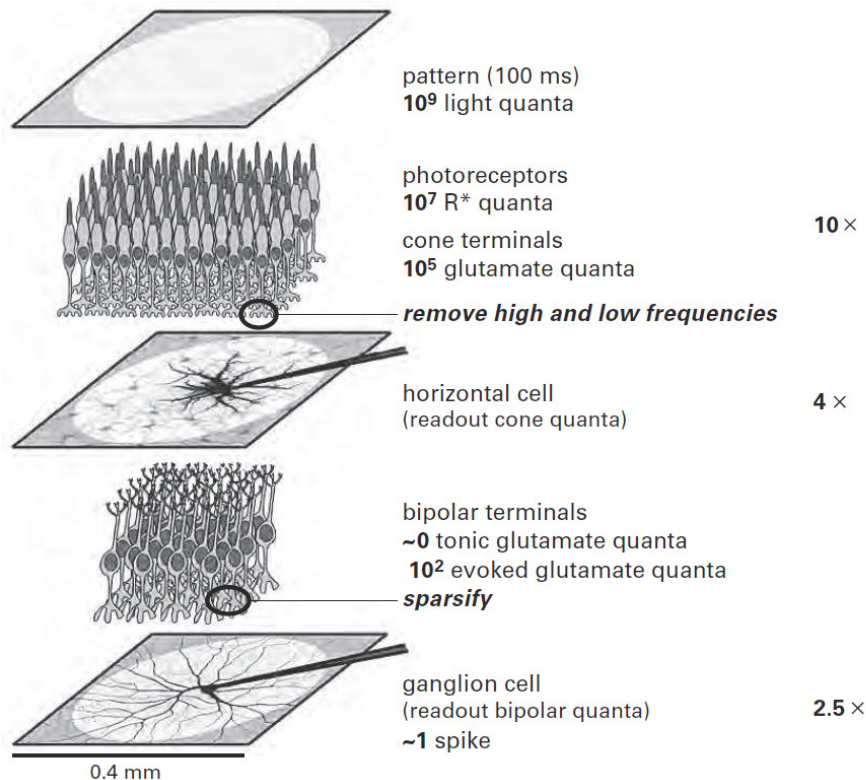
# Patrones arquitectónicos

## Capas



### No sólo en software...

#### Retina



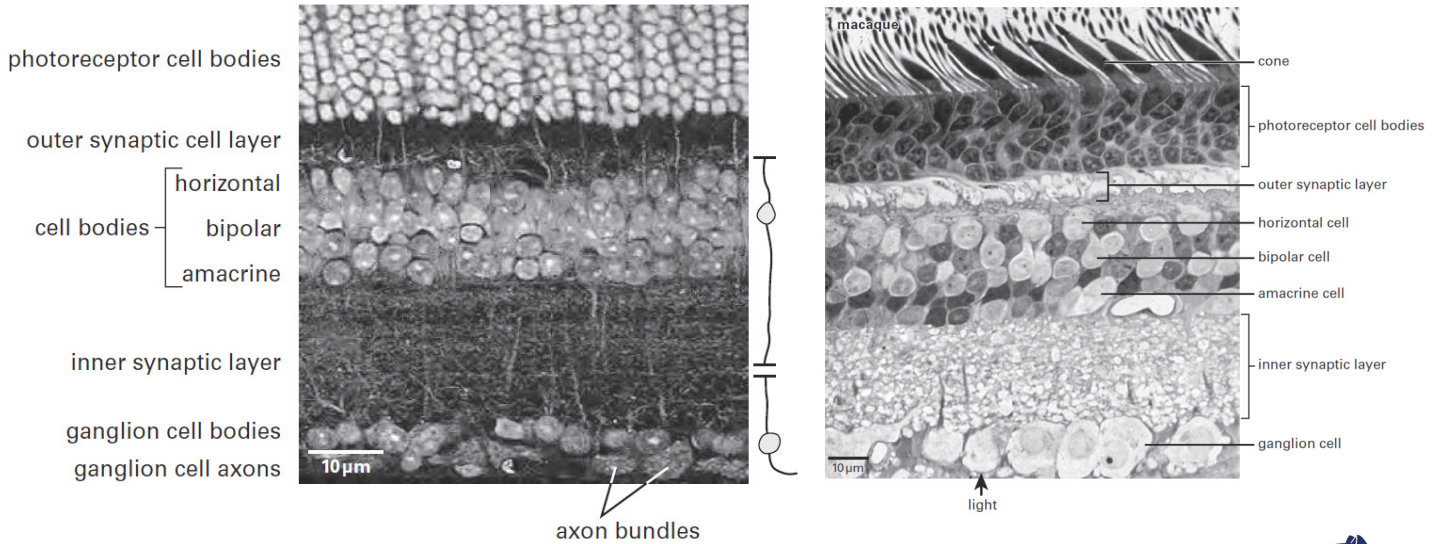
# Patrones arquitectónicos

## Capas



### No sólo en software...

#### Retina



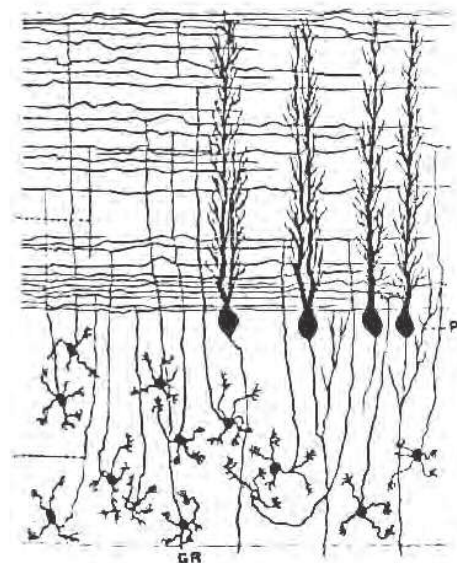
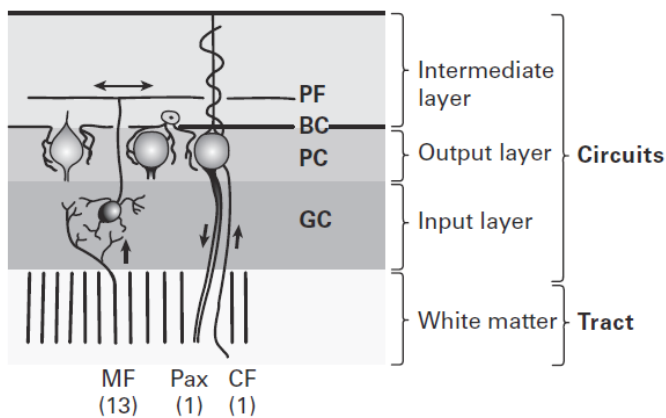
# Patrones arquitectónicos

## Capas



### No sólo en software...

#### Córtex cerebelar



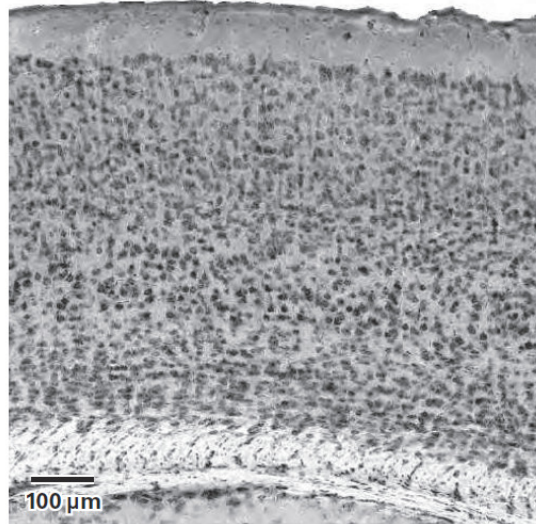
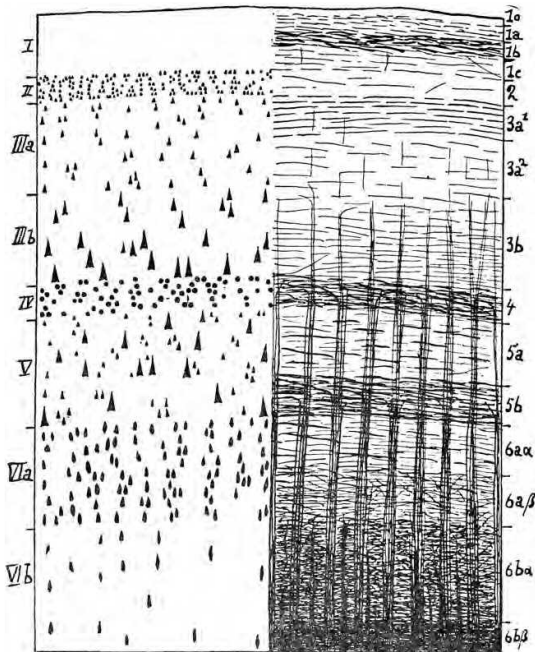
# Patrones arquitectónicos

## Capas



No sólo en software...

Córtex cerebral



I  
II  
III  
IV  
V  
VI  
white matter



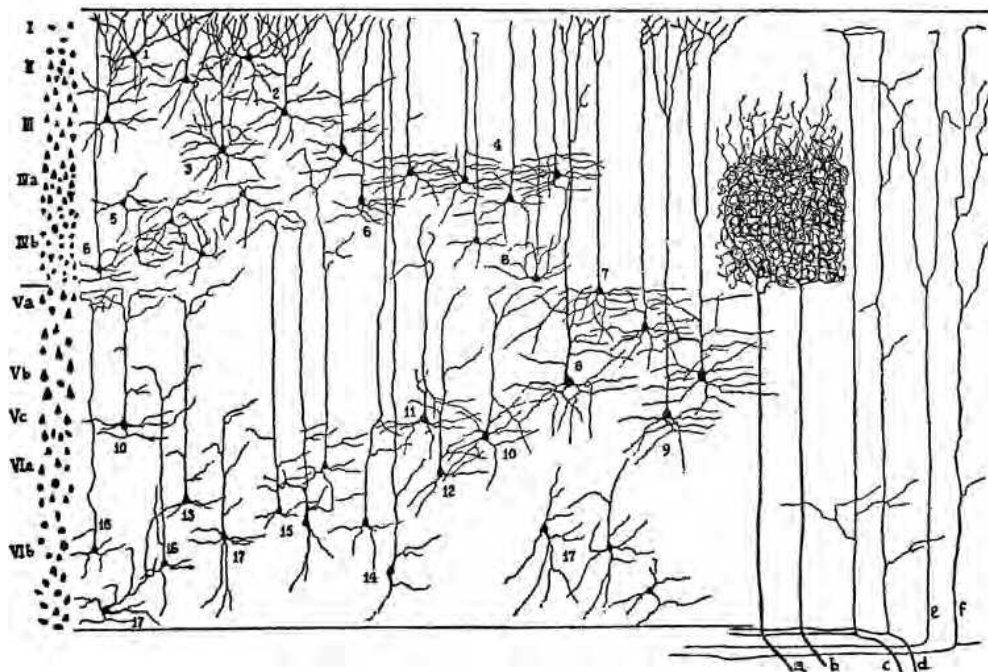
# Patrones arquitectónicos

## Capas



No sólo en software...

Córtex cerebral

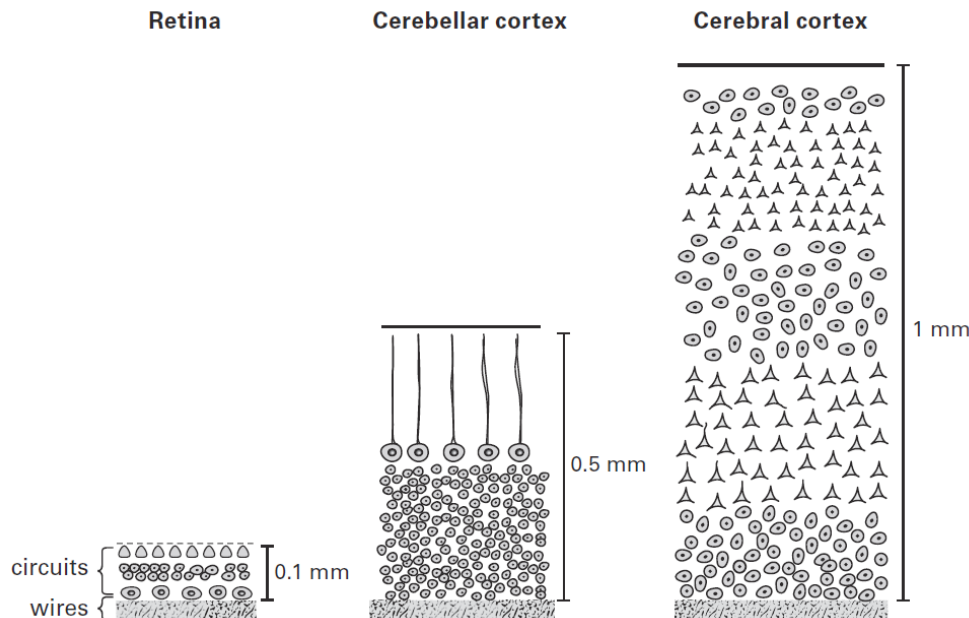


# Patrones arquitectónicos

## Capas



No sólo en software...



# Patrones arquitectónicos

## Capas



### Diseño arquitectónico

Refinamiento progresivo

- Definir el criterio de abstracción (que nos permitirá agrupar tareas en capas).
- Determinar el número de niveles de abstracción (cada nivel de abstracción corresponde a una capa).
- Nombrar las capas.
- Asignar responsabilidades a cada una de ellas.
- Especificar los servicios que ofrece cada capa.

NOTA:

Un enfoque ascendente [bottom-up] o "yo-yo" puede resultar más adecuado...







### Diseño detallado

- Diseñar la interfaz de cada capa.
- Diseñar la estructura de cada capa.
- Diseñar el mecanismo de comunicación entre capas (p.ej. "push model" vs. "pull model").
- Desacoplar capas adyacentes.
- Diseñar un mecanismo de gestión de errores.



### Implementación

```
class L1Provider {
public:
    virtual void L1Service() = 0;
};
class L2Provider {
public:
    virtual void L2Service() = 0;
    void setLowerLayer(L1Provider *l1) {level1 = l1;}
protected:
    L1Provider *level1;
};
class L3Provider {
public:
    virtual void L3Service() = 0;
    void setLowerLayer(L2Provider *l2) {level2 = l2;}
protected:
    L2Provider *level2;
};
```





### Implementación

```
class DataLink : public L1Provider {
public:
    virtual void L1Service() {
        cout << "L1Service doing its job" << endl;
    };
class Transport : public L2Provider {
public:
    virtual void L2Service() {
        cout << "L2Service starting its job" << endl;
        level1->L1Service();
        cout << "L2Service finishing its job" << endl;
    };
class Session : public L3Provider {
public:
    virtual void L3Service() {
        cout << "L3Service starting its job" << endl;
        level2->L2Service();
        cout << "L3Service finishing its job" << endl;
    };
};
```



### Implementación

```
DataLink dataLink;
Transport transport;
Session session;

transport.setLowerLayer(&dataLink);
session.setLowerLayer(&transport);

session.L3Service();
```

```
L3Service starting its job
L2Service starting its job
L1Service doing its job
L2Service finishing its job
L3Service finishing its job
```





### Ventajas

- Reutilización de capas.
- Estandarización de tareas e interfaces.
- Dependencias locales.
- Implementaciones intercambiables.



### Consecuencias

- **Cambios en cascada:**  
Un cambio local en una capa puede afectar a otras (es importante proteger las capas superiores de posibles cambios en las capas inferiores).
- **Eficiencia:**  
La descomposición en capas suele ser menos eficiente que una implementación monolítica equivalente.
- **Trabajo duplicado:**  
Determinadas tareas pueden que se realicen de forma innecesaria (p.ej. comprobación de errores).



# Patrones arquitectónicos

## Capas



### Variante: Sistema basado en capas "relajado"

Cada capa puede utilizar los servicios de todas las capas que quedan debajo de ella.

- Mayor flexibilidad y rendimiento.
- Menor mantenibilidad.

p.ej. Sistemas operativos UNIX

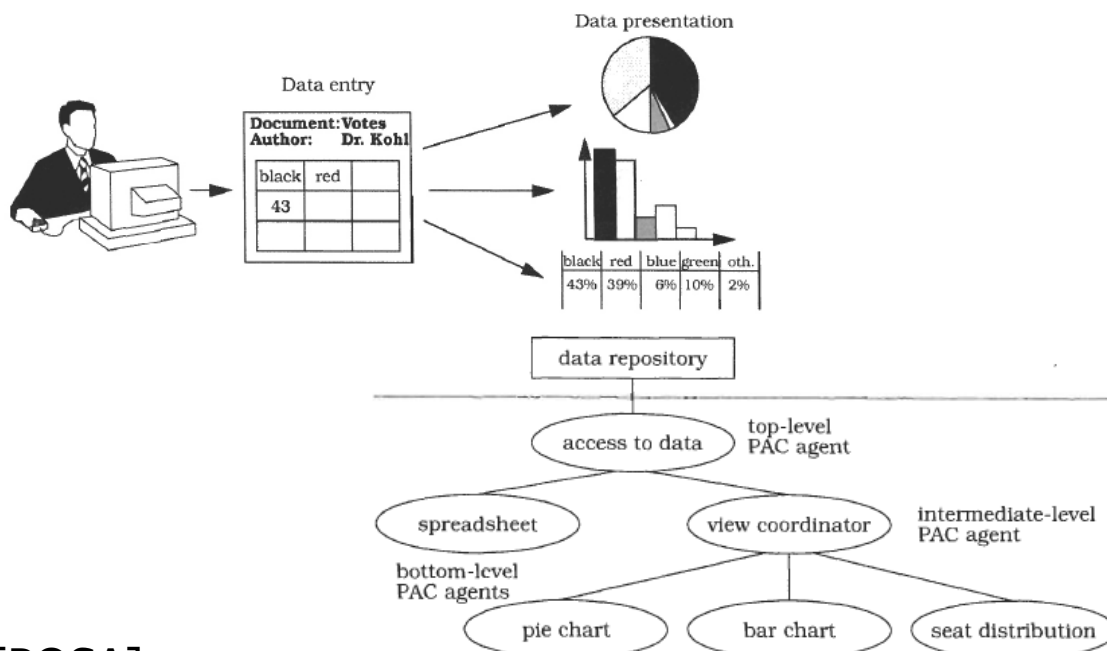


# Patrones arquitectónicos

## Capas



### Variante: PAC [Presentation-Abstraction-Control]



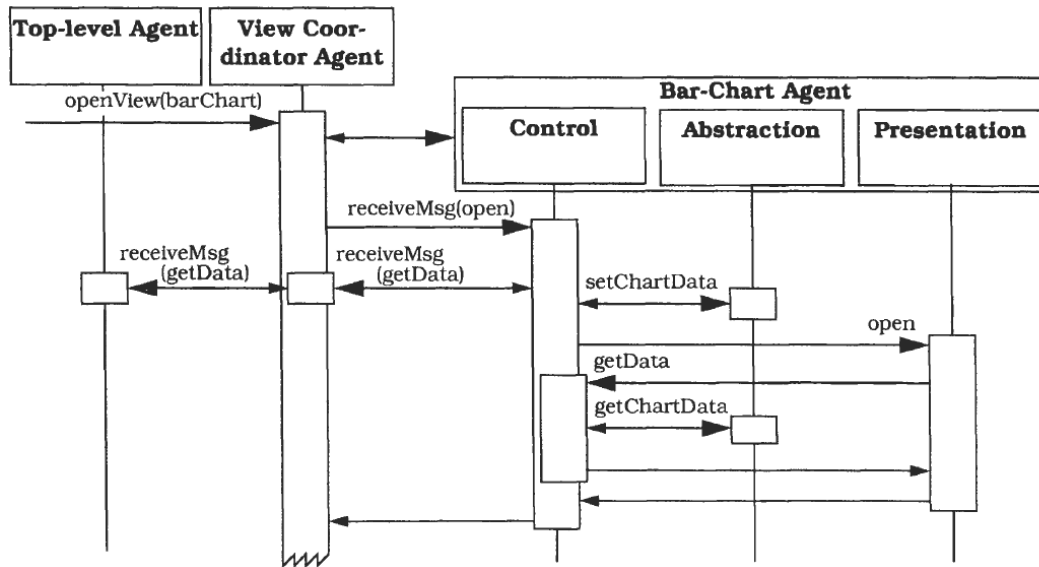
[POSA]





### Variante:

### PAC [Presentation-Abstraction-Control]



[POSA]



### Variante:

### Microkernel

Separa el núcleo funcional mínimo del resto (para poder adaptarse a requisitos cambiantes).

Class	Collaborators
Microkernel	<ul style="list-style-type: none"> <li>Internal Server</li> </ul>
<b>Responsibility</b> <ul style="list-style-type: none"> <li>Provides core mechanisms.</li> <li>Offers communication facilities.</li> <li>Encapsulates system dependencies.</li> <li>Manages and controls resources.</li> </ul>	

[POSA]



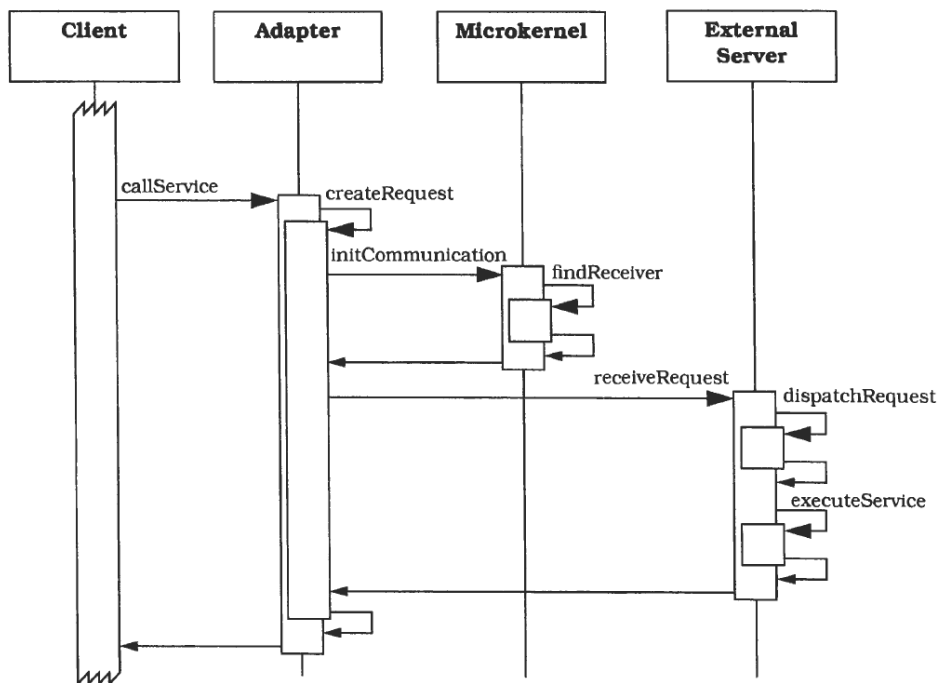


### Variante: Microkernel

<p><b>Class</b> Internal Server</p> <p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Implements additional services.</li> <li>• Encapsulates some system specifics.</li> </ul>	<p><b>Collaborators</b></p> <ul style="list-style-type: none"> <li>• Microkernel</li> </ul>	<p><b>Class</b> External Server</p> <p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Provides programming interfaces for its clients.</li> </ul>	<p><b>Collaborators</b></p> <ul style="list-style-type: none"> <li>• Microkernel</li> </ul>
<p><b>Class</b> Client</p> <p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Represents an application.</li> </ul>	<p><b>Collaborators</b></p> <ul style="list-style-type: none"> <li>• Adapter</li> </ul>	<p><b>Class</b> Adapter</p> <p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Hides system dependencies such as communication facilities from the client.</li> <li>• Invokes methods of external servers on behalf of clients.</li> </ul>	<p><b>Collaborators</b></p> <ul style="list-style-type: none"> <li>• External Server</li> <li>• Microkernel</li> </ul>



### Variante: Microkernel



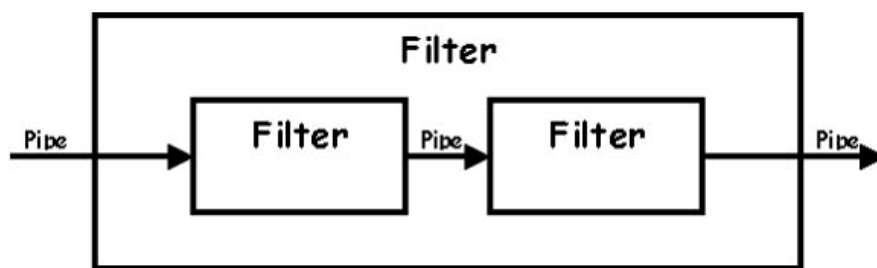
# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

Para sistemas que tengan que procesar flujos de datos:

- Cada etapa de procesamiento es un filtro.
- Los datos pasan de un filtro a otro mediante canales.

Recombinar filtros sirve para construir familias completas de sistemas relacionados:



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

<b>Class</b> Filter	<b>Collaborators</b> • Pipe	<b>Class</b> Pipe	<b>Collaborators</b> • Data Source • Data Sink • Filter
<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Gets input data.</li> <li>• Performs a function on its input data.</li> <li>• Supplies output data.</li> </ul>		<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Transfers data.</li> <li>• Buffers data.</li> <li>• Synchronizes active neighbors.</li> </ul>	

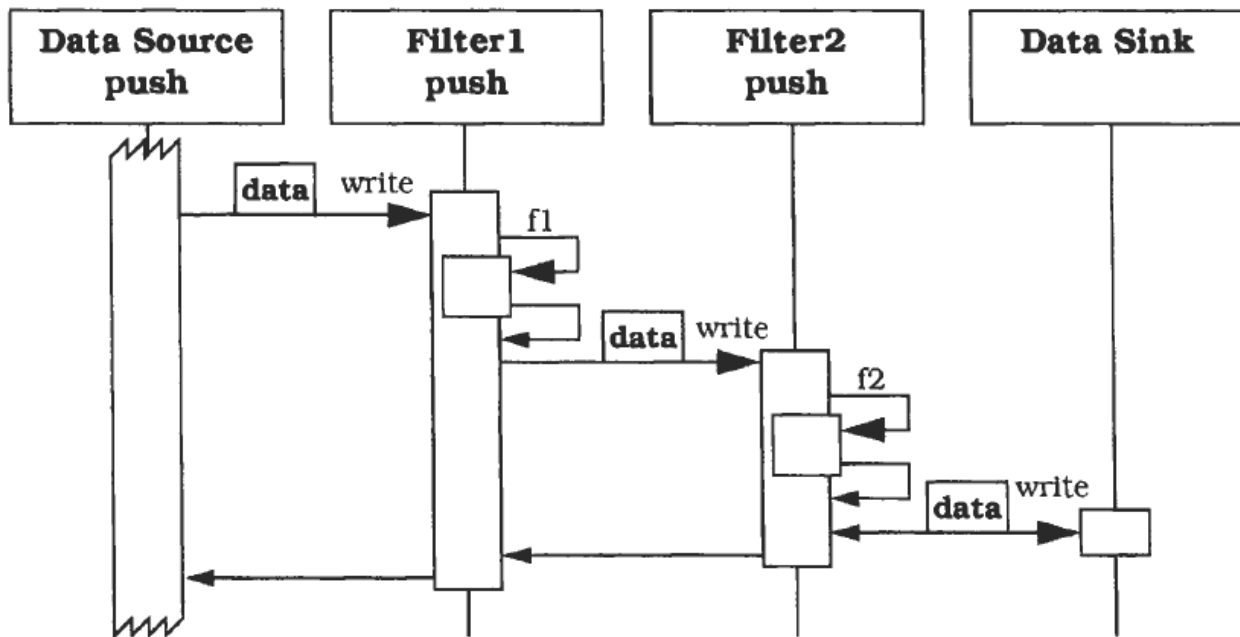
<b>Class</b> Data Source	<b>Collaborators</b> • Pipe	<b>Class</b> Data Sink	<b>Collaborators</b> • Pipe
<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Delivers input to processing pipeline.</li> </ul>		<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Consumes output.</li> </ul>	



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

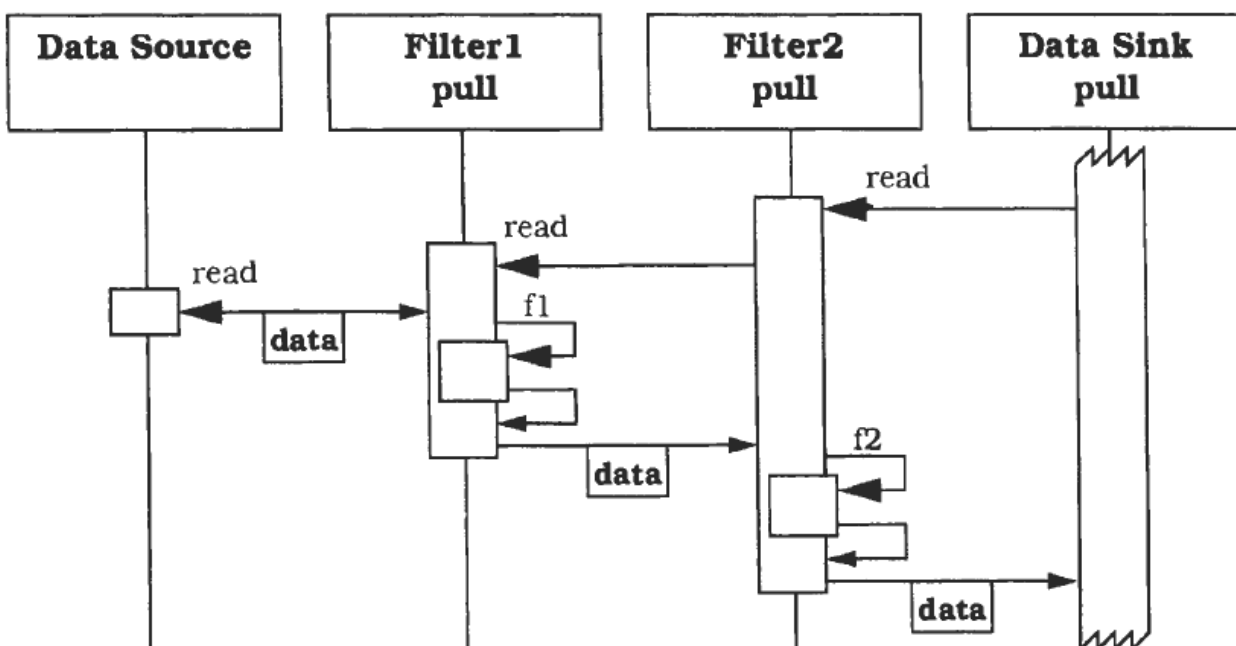
### Push pipeline



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Pull pipeline

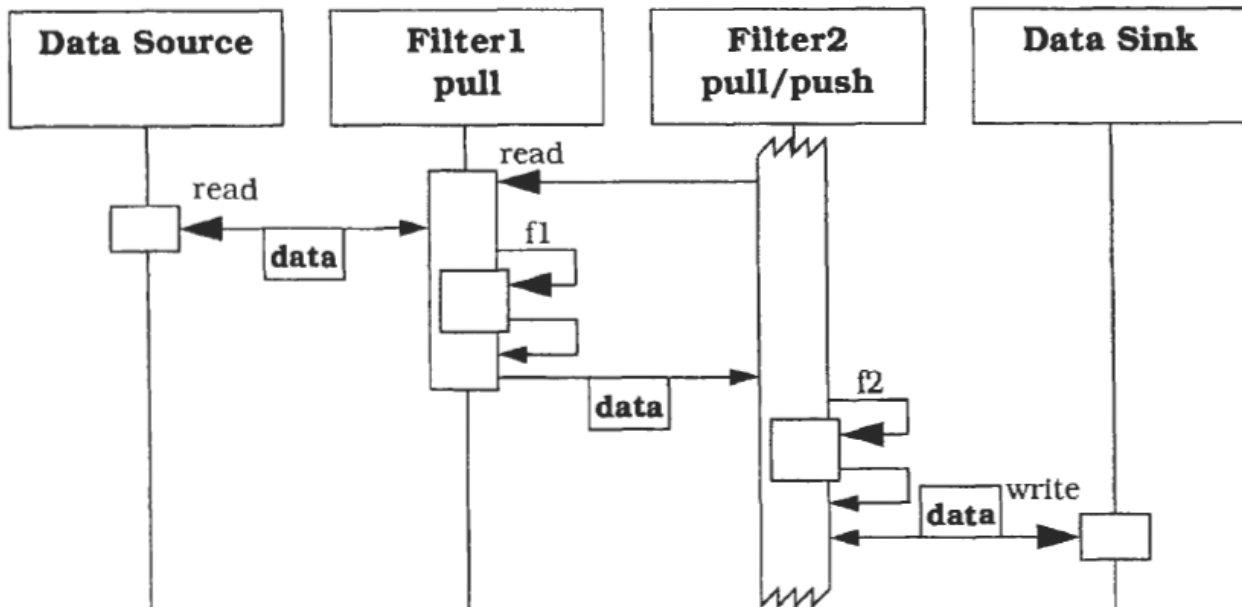




# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

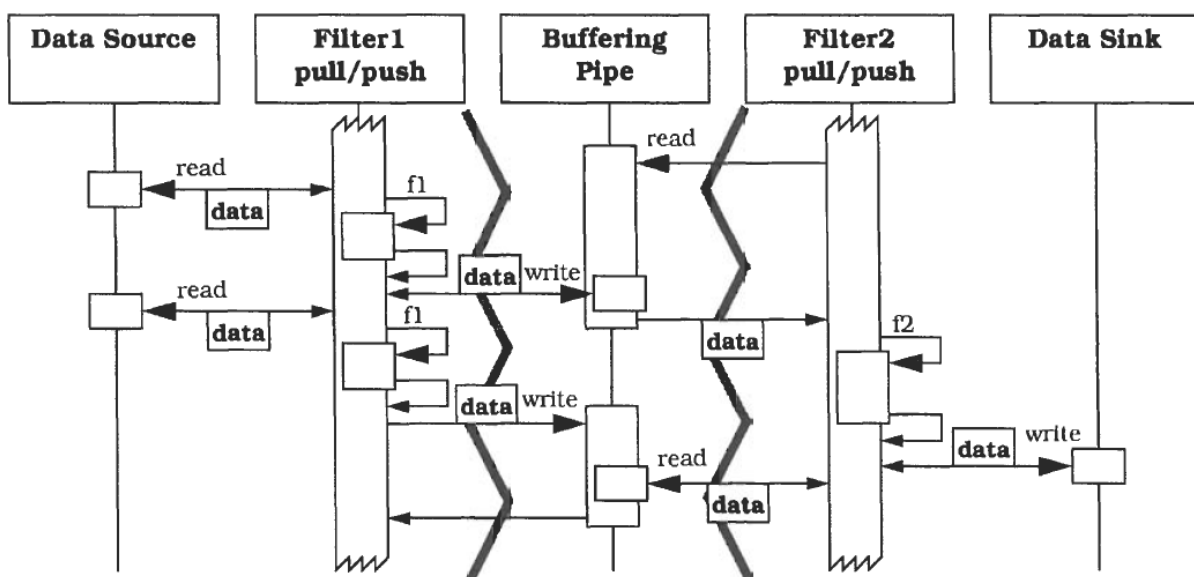
### Mixed pipeline



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Pipes & filter system

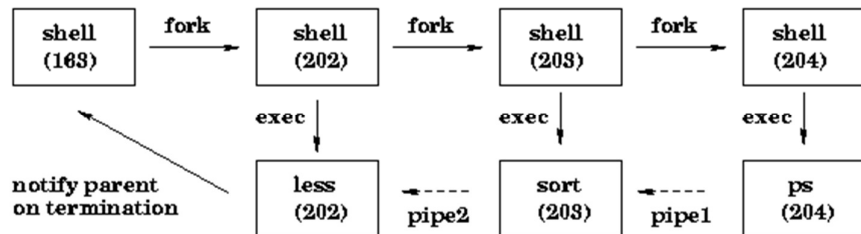
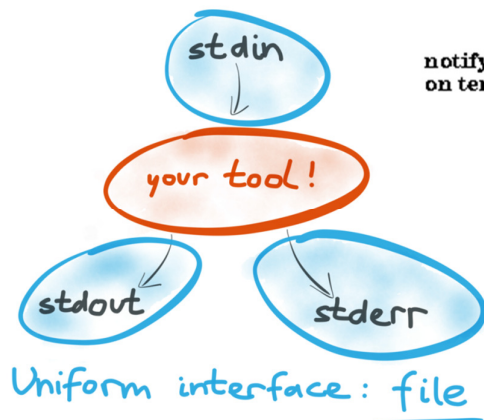


# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Herramientas UNIX

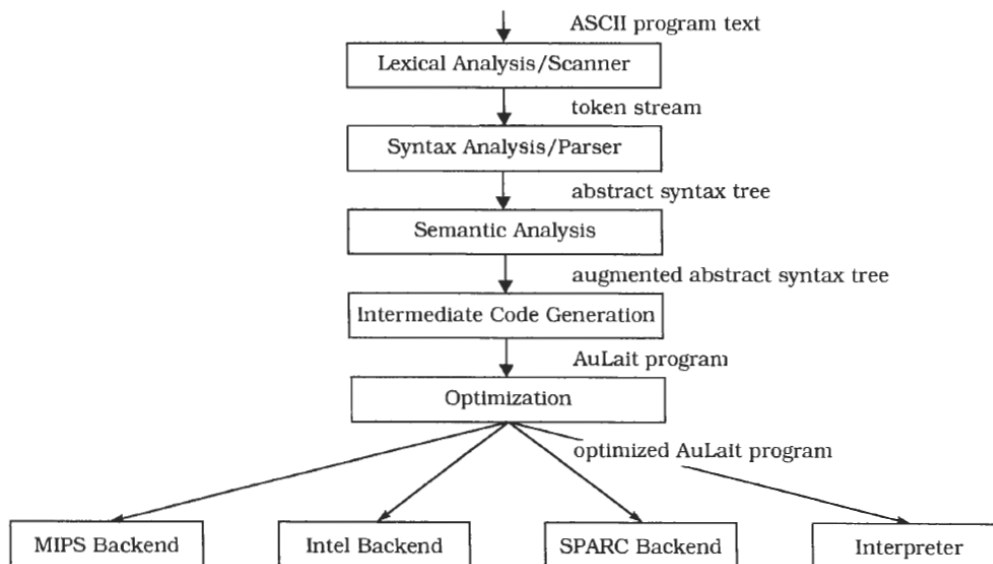
ps | sort | less



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

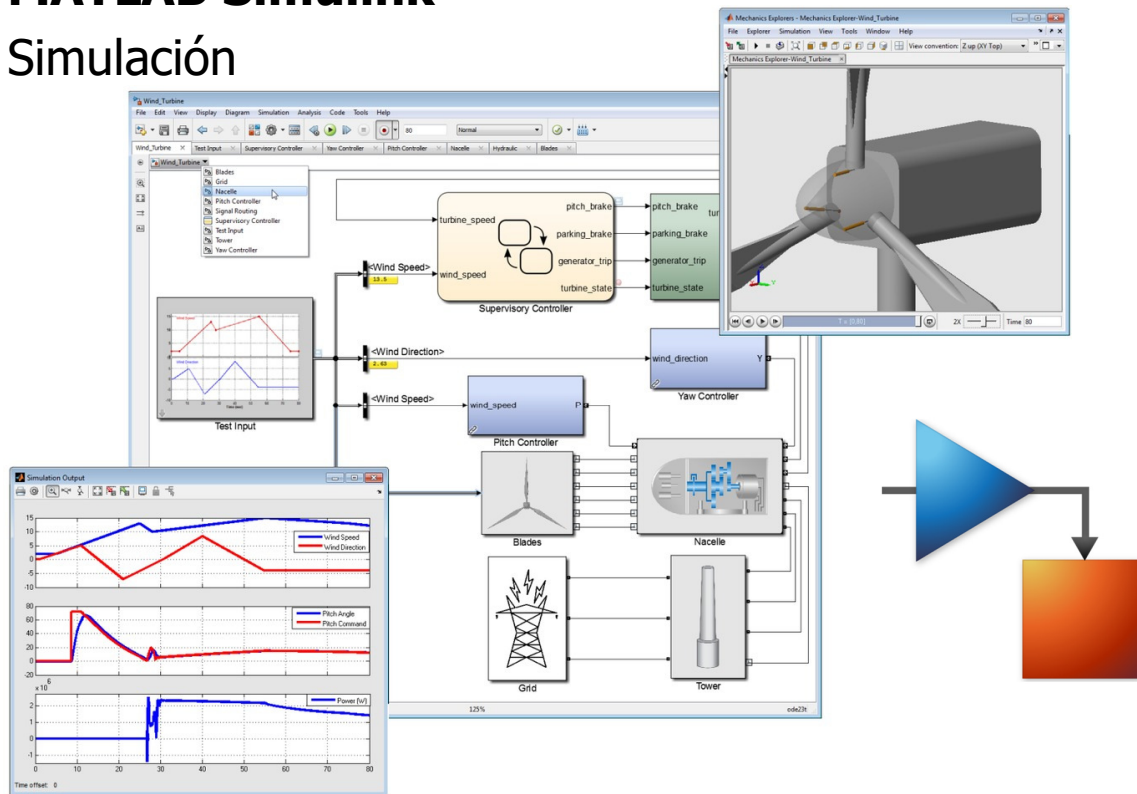
### Intérpretes y compiladores



# Patrones arquitectónicos Flujos de datos [pipes & filters]

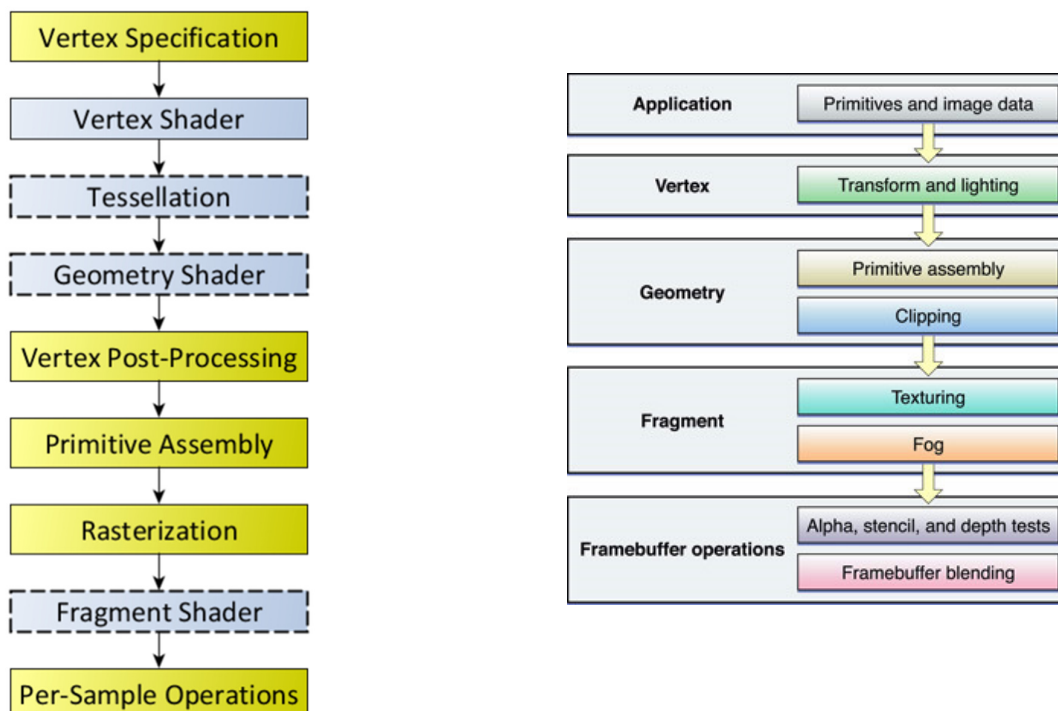
## MATLAB Simulink

### Simulación



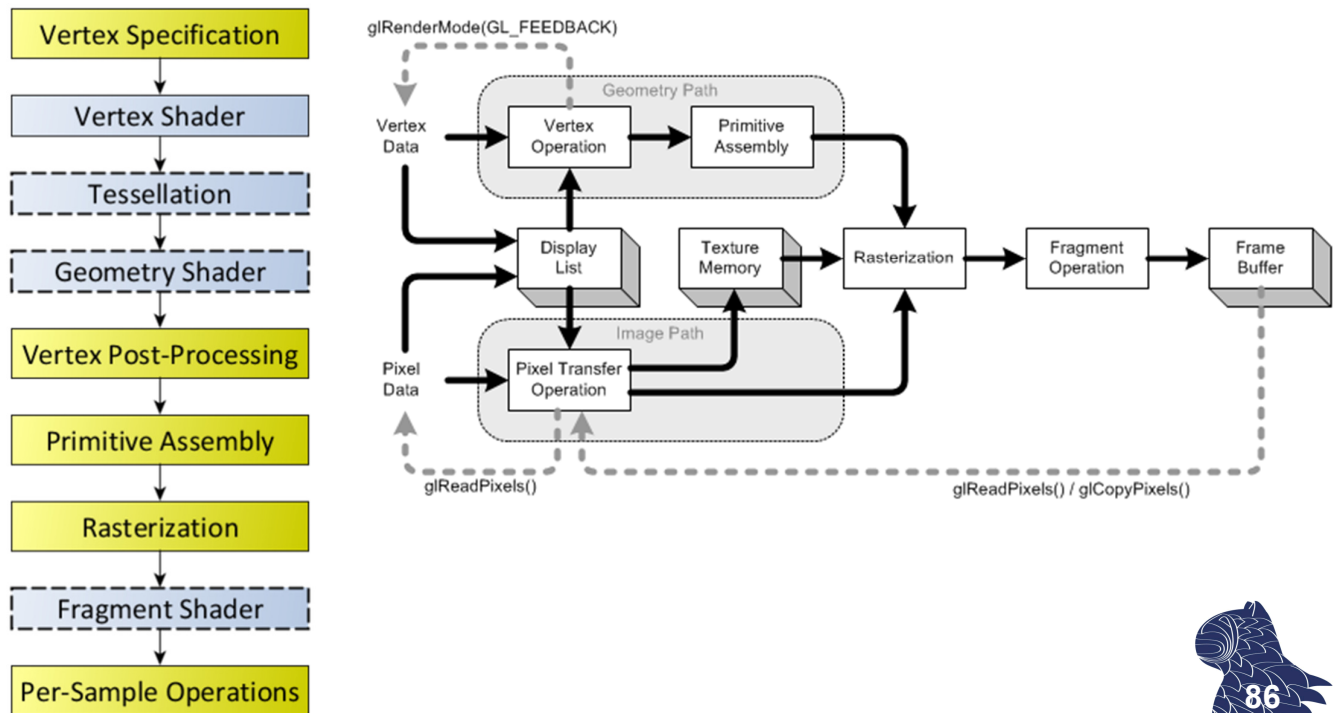
# Patrones arquitectónicos Flujos de datos [pipes & filters]

## OpenGL rendering pipeline



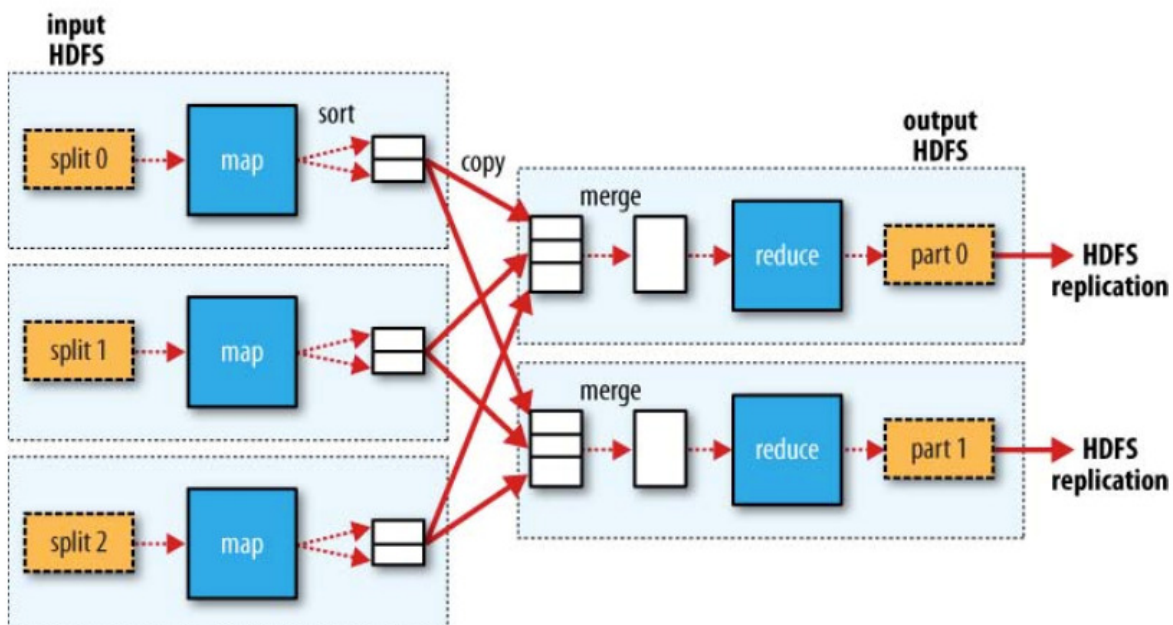
# Patrones arquitectónicos Flujos de datos [pipes & filters]

## OpenGL rendering pipeline



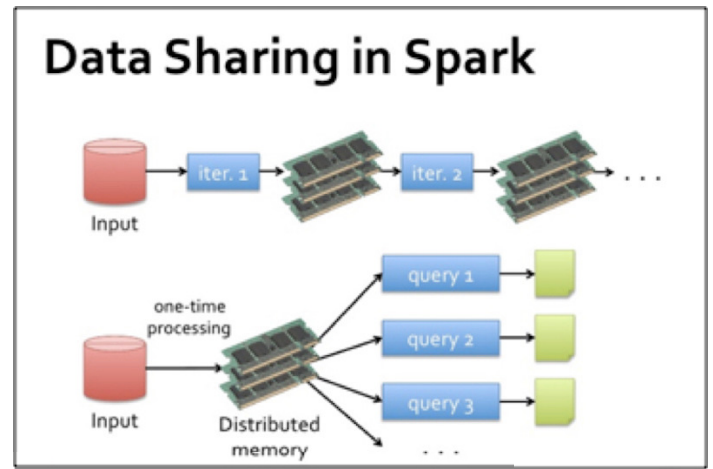
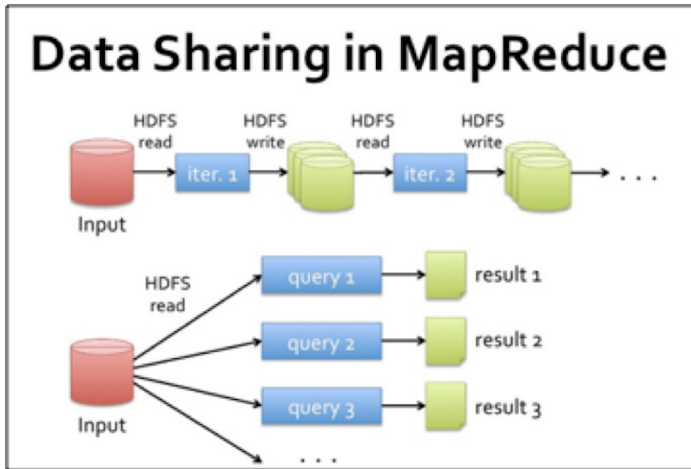
# Patrones arquitectónicos Flujos de datos [pipes & filters]

## MapReduce



# Patrones arquitectónicos Flujos de datos [pipes & filters]

## MapReduce



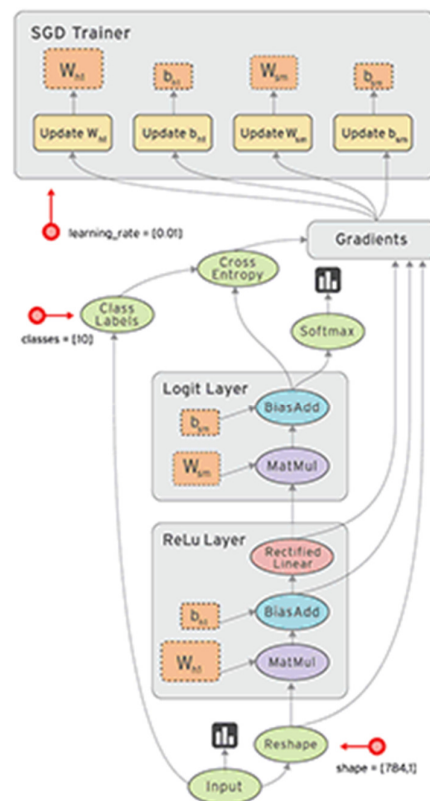
# Patrones arquitectónicos Flujos de datos [pipes & filters]

## Google TensorFlow

<https://www.tensorflow.org/>



Data flow graph

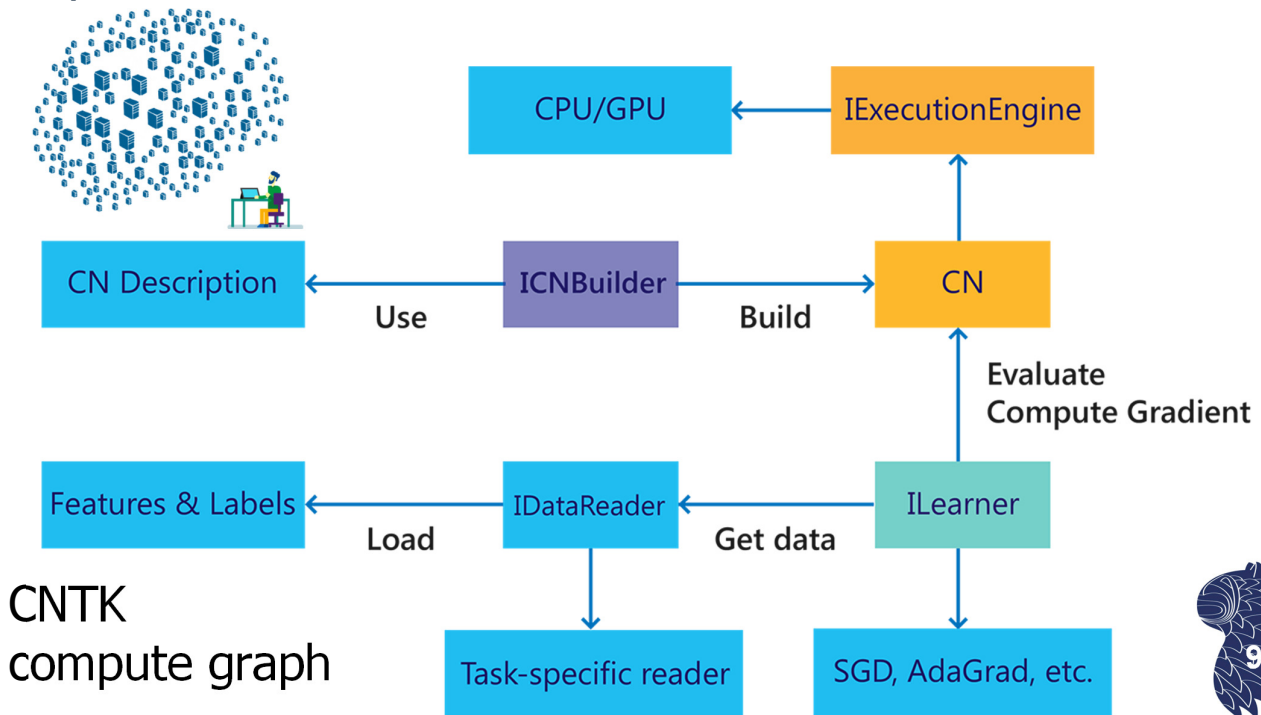


# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Microsoft CNTK [Computational Network Toolkit]

<http://www.cntk.ai/>



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Diseño

- Dividir la tarea en una secuencia de etapas de procesamiento de forma que la salida de cada etapa sólo dependa de las salidas de sus etapas adyacentes.
- Definir el formato de los datos que se transmitirán a través de los canales entre etapas adyacentes (p.ej. ASCII, XML, JSON...).
- Diseñar los filtros  
... y el mecanismo de gestión de errores.



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Implementación de las conexiones

Manual:

- Llamadas directas entre filtros.
- Colas con productores/consumidores.

Utilizando mecanismos del sistema:

- UNIX pipes
- Middleware (colas de mensajes)



# Patrones arquitectónicos

## Flujos de datos [pipes & filters]

### Ventajas

- Flexibilidad:
  - Intercambio de filtros.
  - Recombinación de filtros.
- Reutilización de componentes.
- Ficheros intermedios innecesarios (aunque posibles).
- Prototipado rápido.
- Procesamiento paralelo.



### Consecuencias

- Datos compartidos (compartir muchos datos globales puede ocasionar problemas de rendimiento).
- El coste de transmisión de los datos puede ser significativo en comparación con el tiempo de procesamiento de los datos.
- La implementación en procesos/hebras independientes causa frecuentes cambios de contexto.
- La sincronización detiene los filtros a menudo si los canales de comunicación tienen búferes pequeños.



### Consecuencias

- Algunos filtros puede que requieran consumir todos sus datos de entrada antes de producir salida alguna (p.ej. ordenación) o no estén correctamente implementados (procesamiento incremental).
- Overhead debido a la transformación de los datos: un formato único es más flexible (p.ej. UNIX, XML) pero requiere múltiples conversiones de formato.
- Gestión de errores: El "talón de Aquiles" del patrón.





# Patrones arquitectónicos

## Pizarra



Útil en problemas para los que no se conocen estrategias determinísticas que permitan resolverlos.



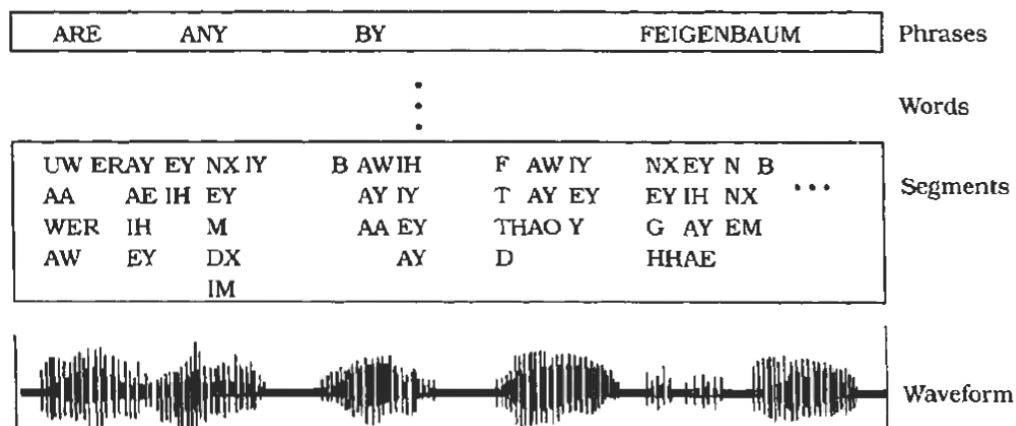
# Patrones arquitectónicos

## Pizarra



### HEARSAY-II

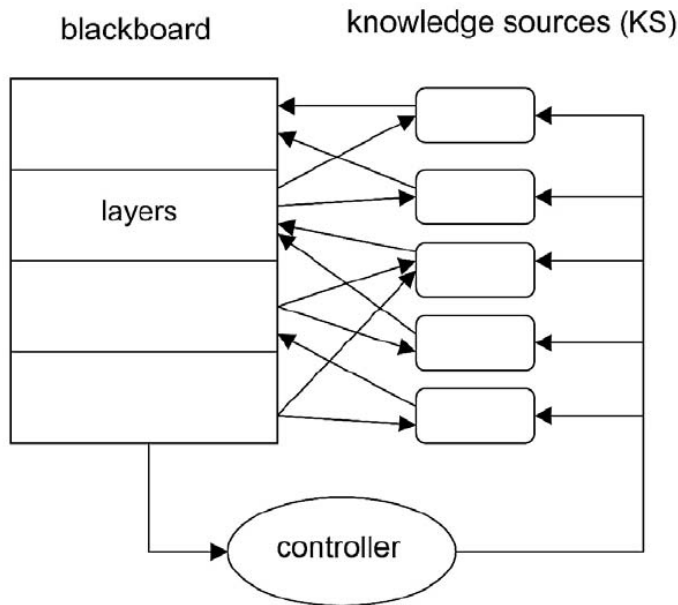
Reconocimiento de voz





## HEARSAY-II

### Reconocimiento de voz

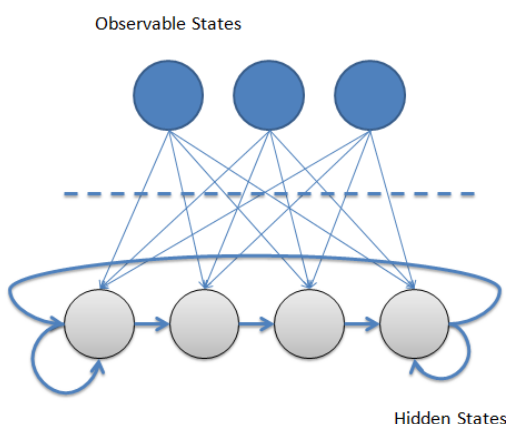


LEVELS	KNOWLEDGE SOURCES
DATA BASE INTERFACE	SEMANT
PHRASE	PARSE, PREDICT, STOP, CONCAT, WORD-SEQ-CTL
WORD-SEQUENCE	WORD-SEQ, WORD-CTL, RPOL
WORD	MOW, VERIFY
SYLLABLE	POM
SEGMENT	SEG
PARAMETER	



## Evolución de los sistemas de reconocimiento de voz

Los sistemas basados en pizarra eran populares antes del "invierno de la IA": Estudios sobre complejidad computacional en los años 70 (problemas NP-difíciles).



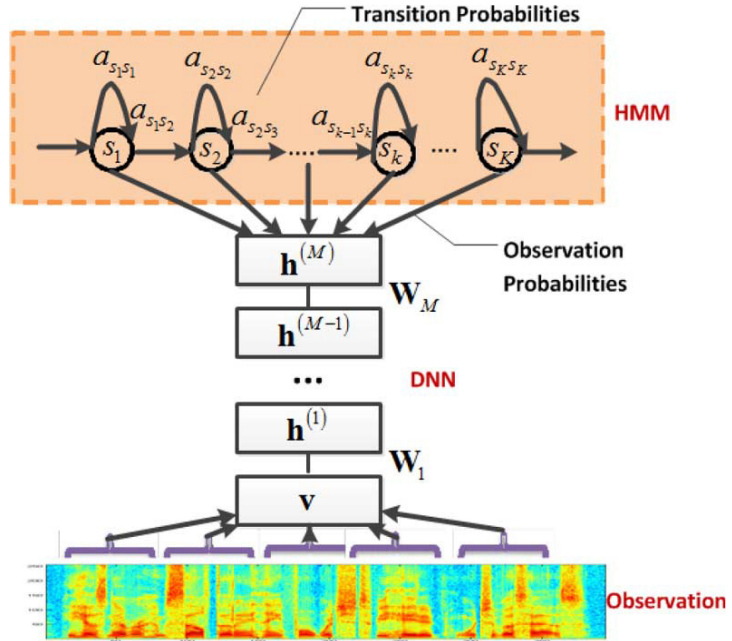
Técnicas estadísticas como los modelos ocultos de Markov [HMM] reemplazaron a los sistemas simbólicos como Hearsay-II.





## Evolución de los sistemas de reconocimiento de voz

Actualmente, se siguen utilizando HMMs cuyos parámetros se ajustan con redes neuronales [deep learning].

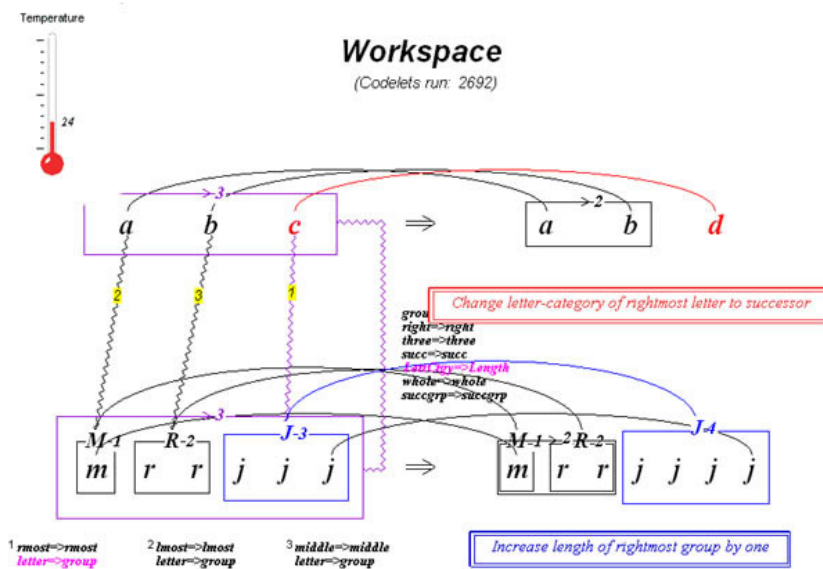


Microsoft  
**Research**



## Copycat

Razonamiento mediante analogías



Here, Metacat describes why a pre-solved puzzle answer makes sense. It says that in the first pair, the change is "last letter to its successor", and in the second it's "length of last group +1". The concept "successor" slips to "length +1", and "letter" to "group". The low temperature means it judges this answer "deep".

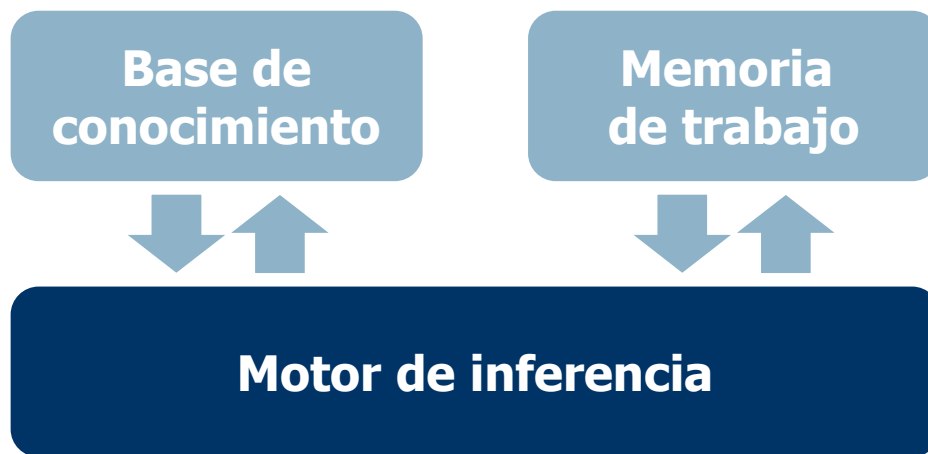


# Patrones arquitectónicos Pizarra



## Sistemas expertos

- Años 70: Primeros sistemas expertos DENDRAL, MYCIN, PROSPECTOR, R1/XCON...
- Años 80: La industria de la I.A. ("boom" de los sistemas expertos)

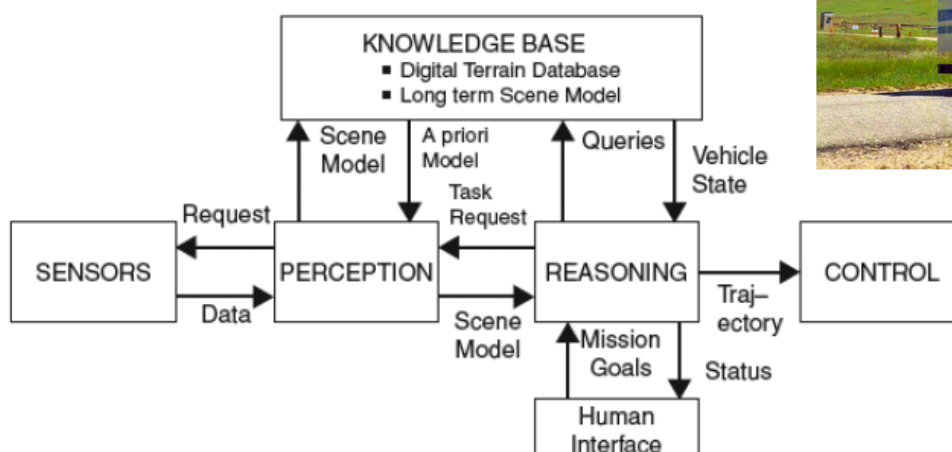


# Patrones arquitectónicos Pizarra



## ALV

Autonomous Land Vehicle



# Patrones arquitectónicos

## Pizarra



### C4ISTAR

Command, Control, Communications, Computers,  
Information/Intelligence, Surveillance, Targeting  
Acquisition and Reconnaissance



Sistemas militares de  
detección y seguimiento de objetos



# Patrones arquitectónicos

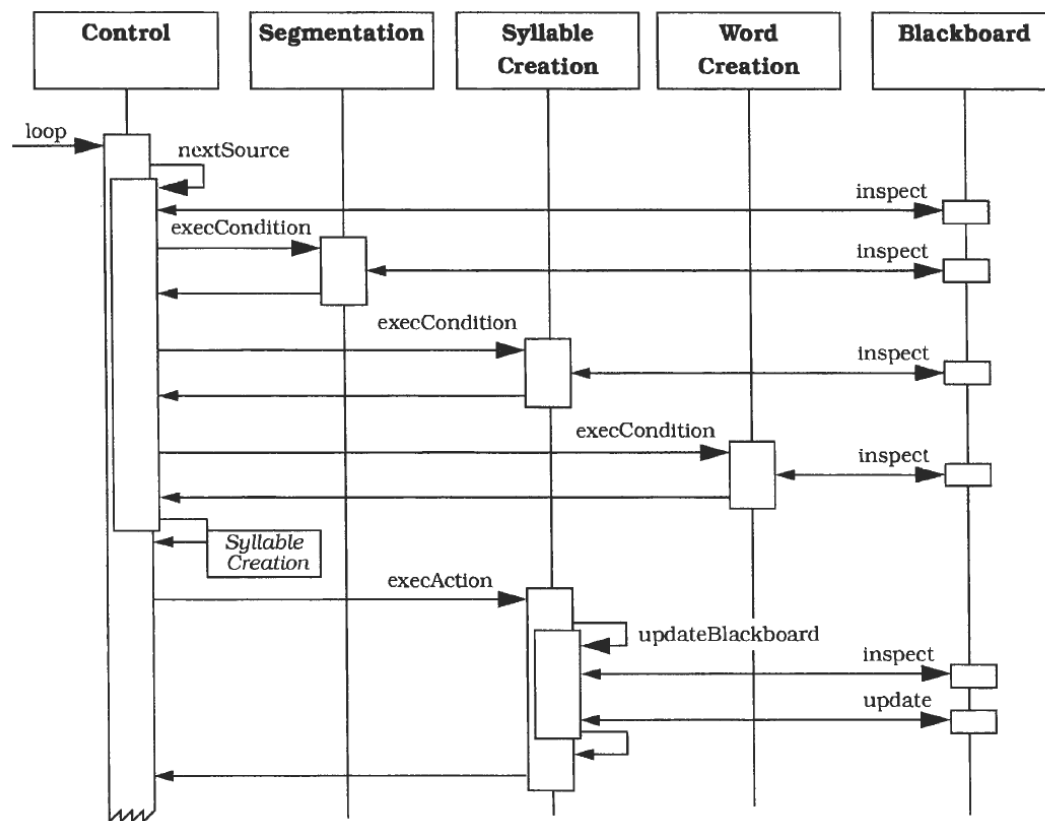
## Pizarra



### Tarjetas CRC

<p><b>Class</b> Blackboard</p>	<p><b>Collaborators</b> -</p>	<p><b>Class</b> Knowledge Source</p>	<p><b>Collaborator</b> • Blackboard</p>
<p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Manages central data</li> </ul>		<p><b>Responsibility</b></p> <ul style="list-style-type: none"> <li>• Evaluates its own applicability</li> <li>• Computes a result</li> <li>• Updates Blackboard</li> </ul>	





## Diseño

- Definir el problema y su espacio de soluciones (puede que a distintos niveles de abstracción: soluciones finales vs. resultados intermedios).
- Dividir el proceso en etapas independientes (transformación de resultados intermedios, predicción y verificación de hipótesis...).
- Diseñar subsistemas especializados para resolver subproblemas concretos.





## Diseño

### Mecanismo de control

- La parte más difícil de diseñar: criterios heurísticos.
- Función: Determinar quién debe realizar cambios sobre la pizarra en cada momento.
- Mecanismos de estimación de la credibilidad (p.ej. verosimilitud [likelihood]) de cada hipótesis.



## Variantes

### ■ Sistemas de producción

p.ej. Business rules engines

[https://en.wikipedia.org/wiki/Business\\_rules\\_engine](https://en.wikipedia.org/wiki/Business_rules_engine)

### ■ Repositorios

(sin mecanismo de control centralizado)

p.ej. Bases de datos

IDEs



# Patrones arquitectónicos

## Pizarra



### Ventajas

- Flexibilidad: Experimentación con diferentes algoritmos y heurísticas de control.
- Reutilización: Componentes reutilizables.
- Tolerancia a fallos y robustez frente a ruido e incertidumbre (los resultados en la pizarra no son más que hipótesis y sólo las hipótesis más sólidas sobreviven)



# Patrones arquitectónicos

## Pizarra



### Consecuencias

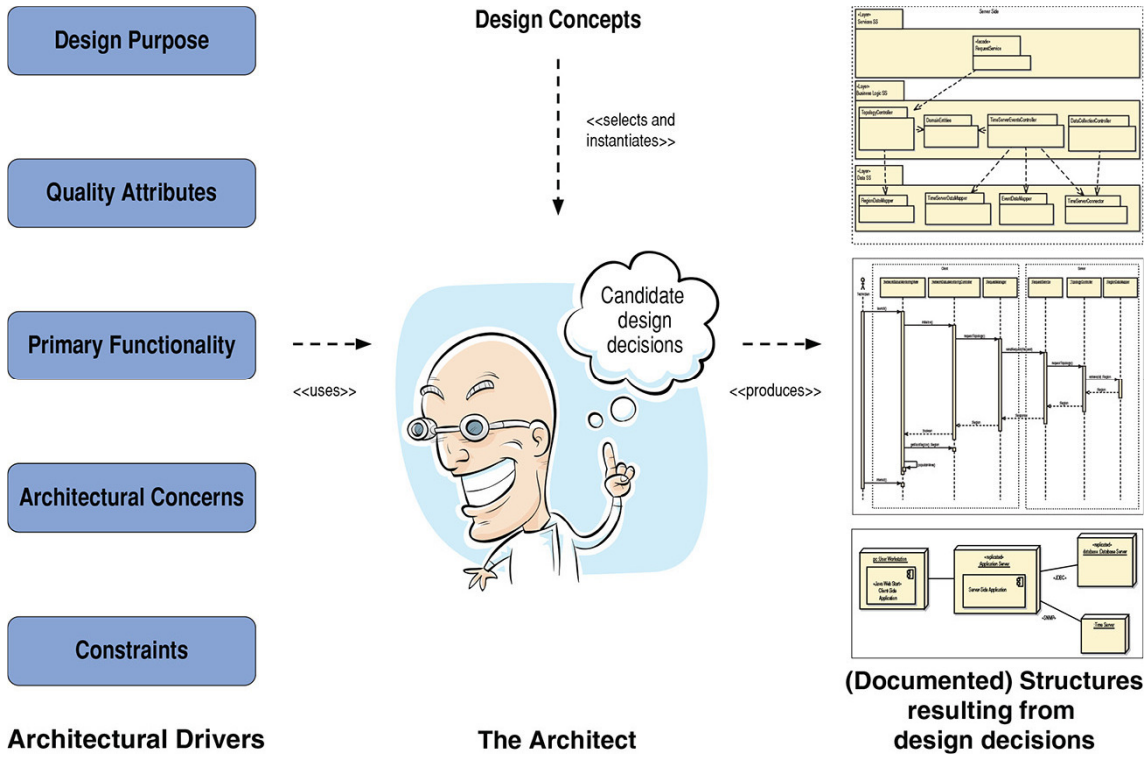
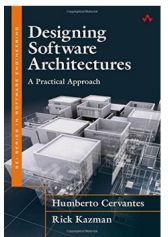
- Difícil de comprobar (pruebas de unidad OK, pruebas de integración problemáticas).
- Sin garantías (no siempre resuelven los problemas).
- Esfuerzo de desarrollo elevado (prueba y error).
- Baja eficiencia (proceso de búsqueda de hipótesis).





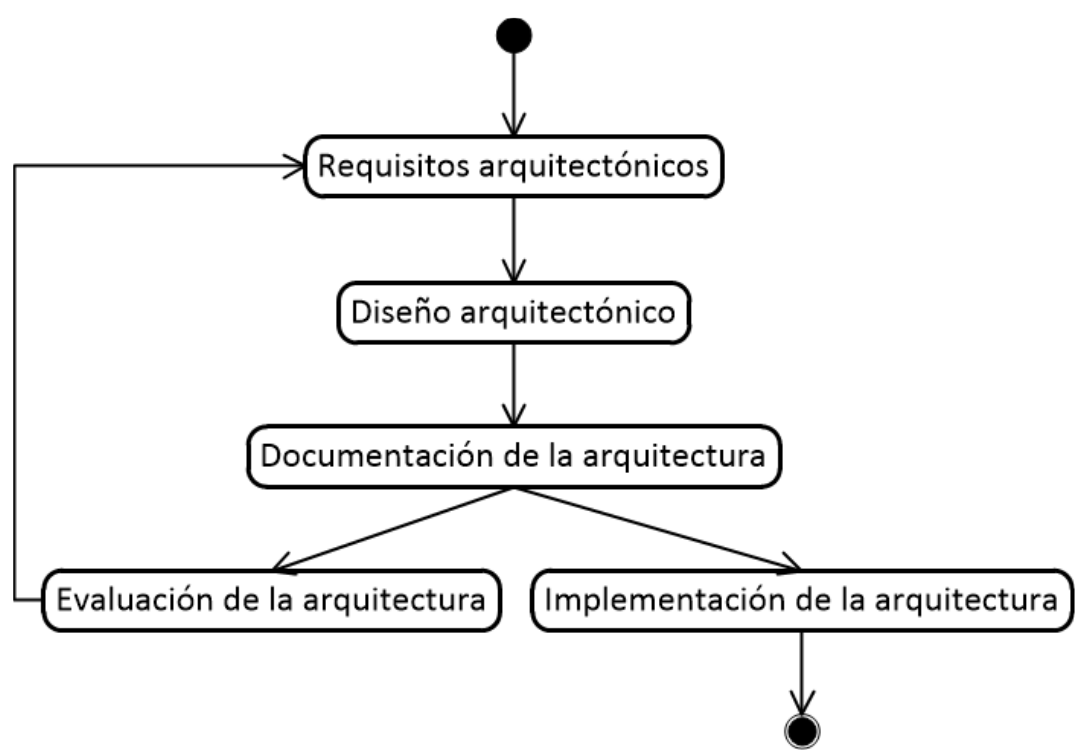
# Diseño arquitectónico

## El proceso de diseño



# Diseño arquitectónico

## El proceso de diseño

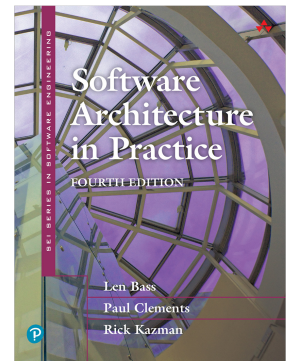
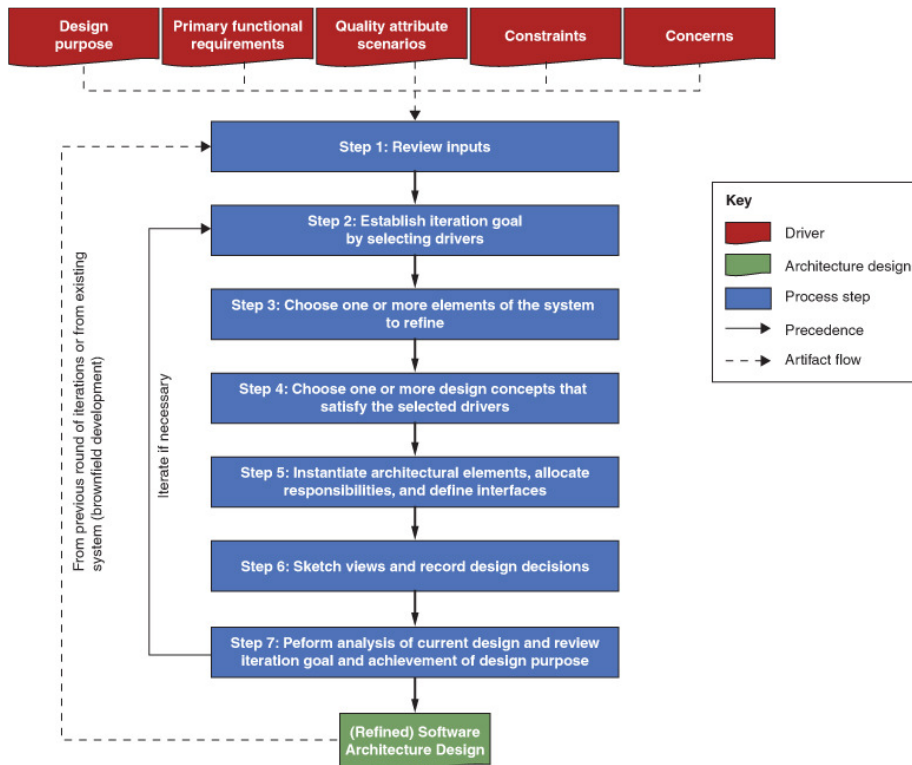


# Diseño arquitectónico

## El proceso de diseño



### ADD [Attribute-Driven Design]



## Validación de la arquitectura



Proceso de validación de las decisiones de diseño:

¿Se tomaron las decisiones correctas?

¿Compromisos adecuados en la resolución de conflictos?

¿Suposiciones acertadas en distintas facetas del sistema?

¿Integridad técnica del sistema?



# Validación de la arquitectura



## Técnicas de validación

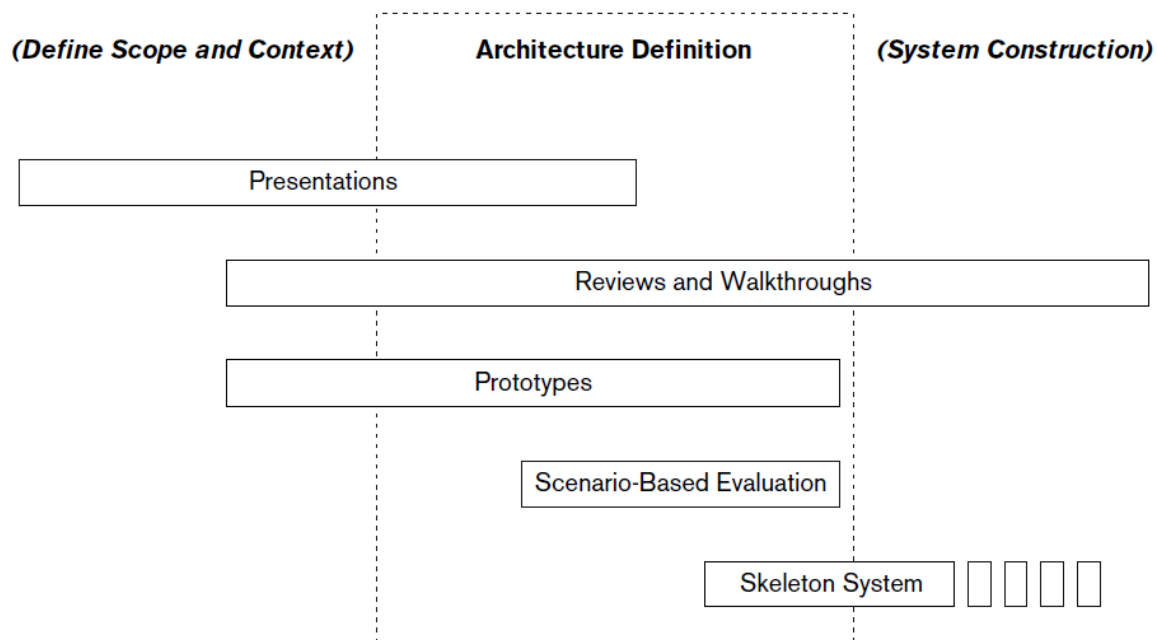
- Presentación (informal) de la arquitectura.
- Revisiones técnicas y recorridos estructurados.
- Evaluación mediante escenarios (p.ej. ATAM & SAAM).
- Prototipos y pruebas de concepto.



# Validación de la arquitectura



## Técnicas de validación





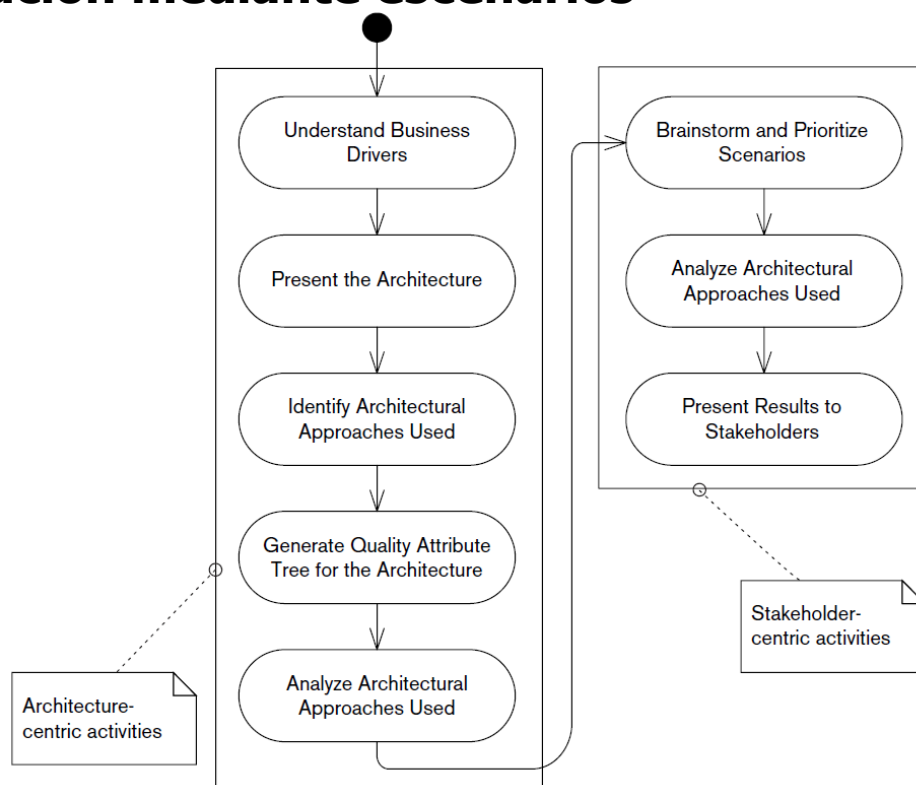
## Evaluación mediante escenarios

- **SAAM [Software Architecture Analysis Method]:**  
Escenarios funcionales para evaluar cómo proporciona el sistema la funcionalidad clave y si resulta fácil adaptarlo frente a cambios probables.
- **ATAM [Architecture Tradeoff Analysis Method]:**  
Extiende SAAM con “quality property scenarios” que evaluar la capacidad del sistema para satisfacer sus requisitos no funcionales.
- **LAE [Lightweight Architecture Evaluation]:**  
ATAM de uso interno para un proyecto.



## Evaluación mediante escenarios

### ATAM

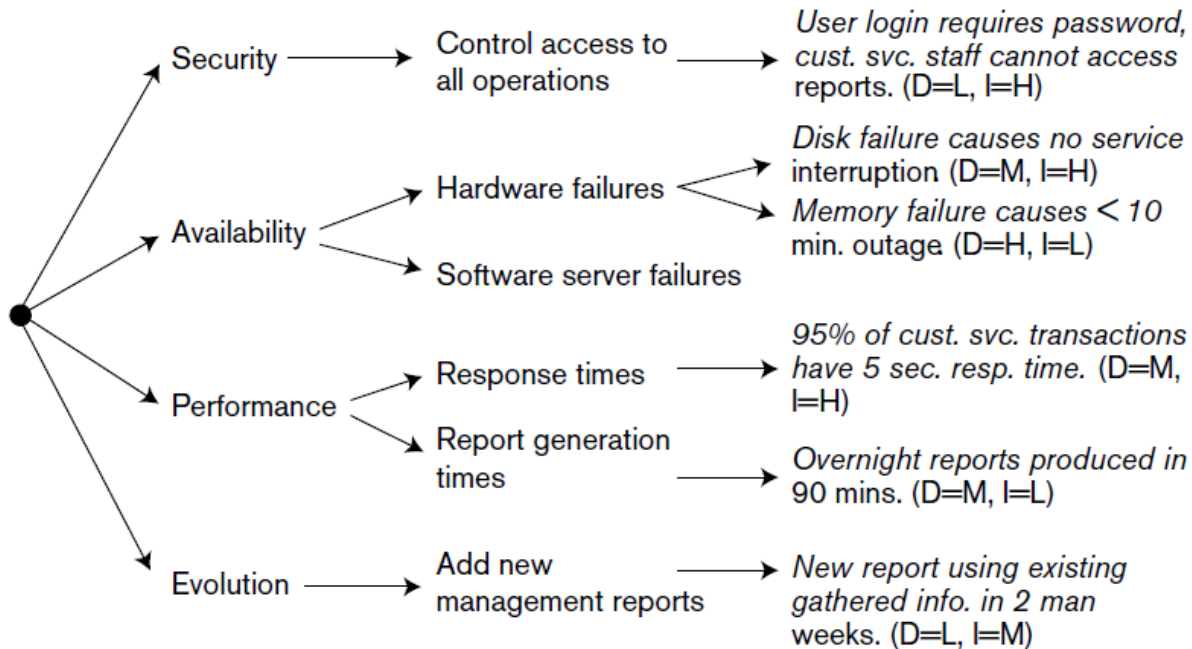


# Validación de la arquitectura



## Evaluación mediante escenarios

### ATAM Quality Attribute Tree, a.k.a. utility tree

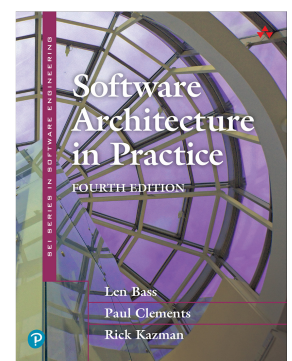
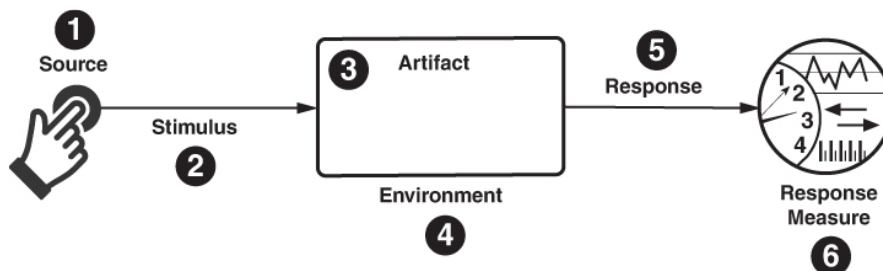


# Validación de la arquitectura



## Evaluación mediante escenarios

### Atributos de calidad [Quality Attributes]



IDEA CLAVE:

Especificación testable, libre de ambigüedades.

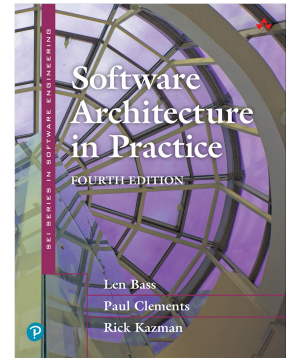
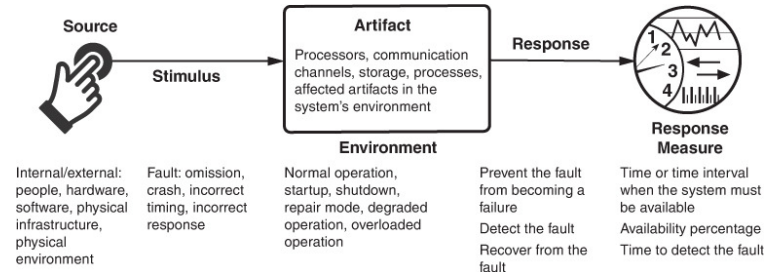


# Validación de la arquitectura

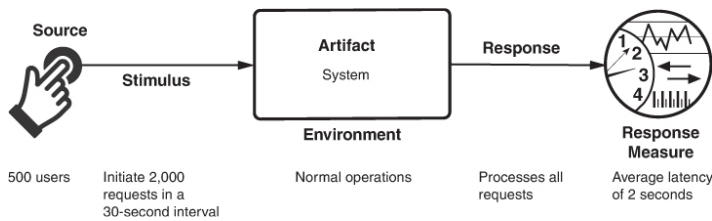


## Evaluación mediante escenarios

### Ejemplos



### DISPONIBILIDAD [AVAILABILITY]



### RENDIMIENTO [PERFORMANCE]

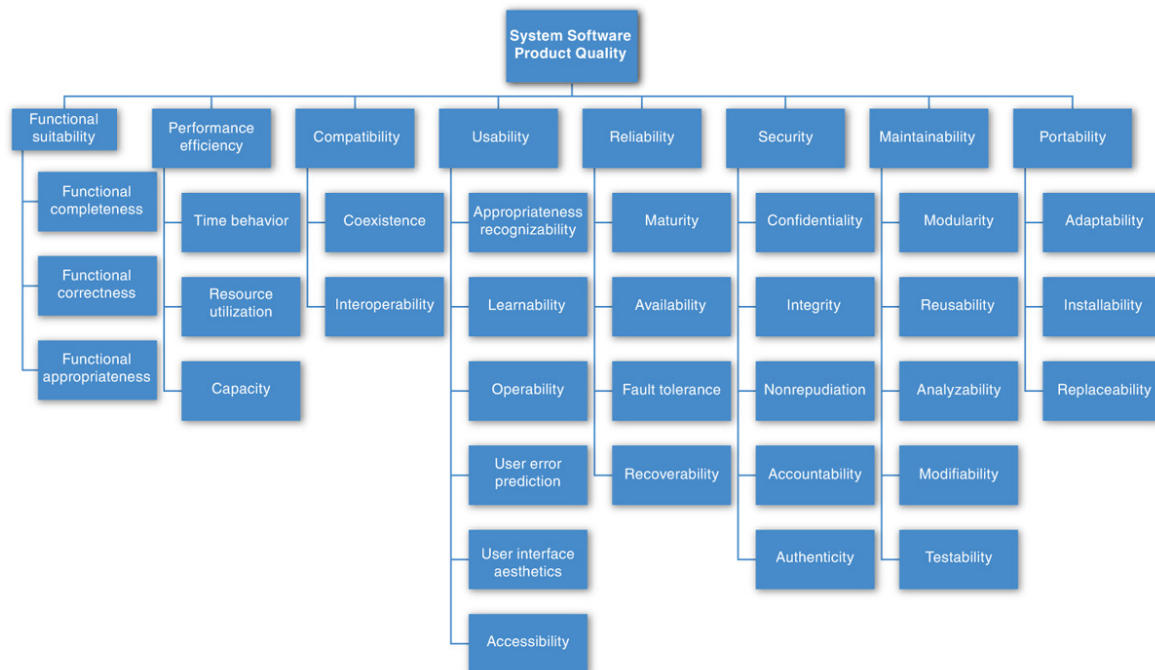


# Validación de la arquitectura



## Evaluación mediante escenarios

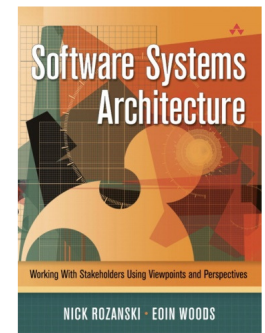
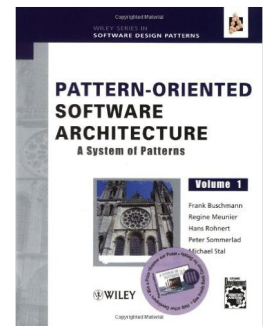
### Atributos de calidad, estándar ISO/IEC FCD 25010



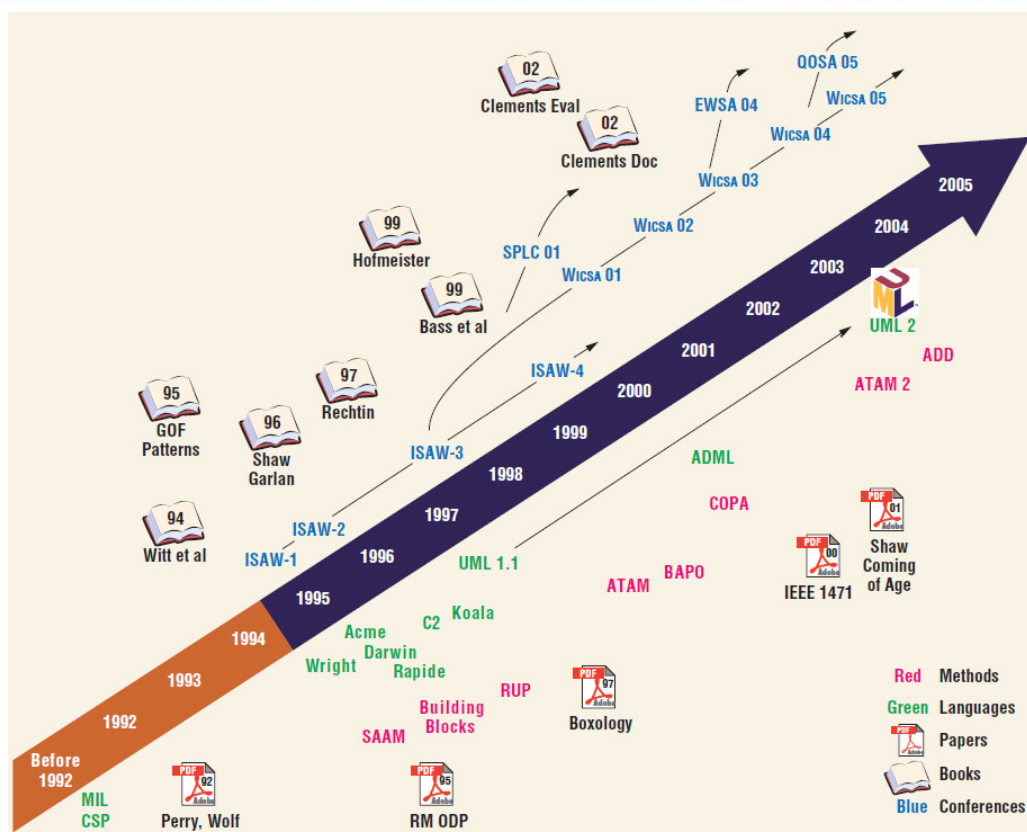
# Bibliografía recomendada



- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad & Michael Stal:  
**Pattern-Oriented Software Architecture. Volume 1: A System of Patterns**  
 Wiley, 1996. ISBN 0471958697
- Nick Rozanski & Eóin Woods:  
**Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives**  
 Addison-Wesley Professional, 2005.  
 ISBN 0321112296



# Bibliografía



The past, present, and future for software architecture  
 IEEE Software 2005

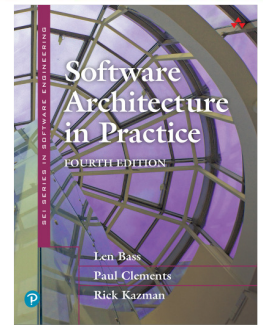


# Apéndice: Conjuntos de vistas

## Software Architecture in Practice

### ESTRUCTURA

1. Vista de módulos (descomposición en unidades de implementación).
2. Vista de componentes y conectores [C&C] (unidades de procesamiento y flujos de datos).
3. Vista de asignación [allocation] ~ despliegue (elementos software en su entorno)



### COMPORTAMIENTO

- Notaciones para "trazas": Casos de uso y diagramas UML (secuencias, comunicación, actividad).
- Notaciones "integrales" [comprehensive]: Diagramas de máquinas de estados.

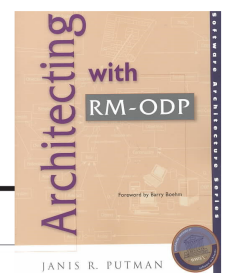


# Apéndice: Conjuntos de vistas

## RM-ODP

### Reference Model for Open Distributed Processing

Viewpoint	Definition
Enterprise	Defines the context for the system and allows capture and organization of requirements.
Information	Describes the information required by the system using static, invariant, and dynamic schemas.
Computational	Contains an object-oriented model of the functional structure of the system, with a particular focus on interfaces and interactions.
Engineering	Describes the systems infrastructure required to implement the desired distribution of the system's elements. This description is performed using a specific reference model.
Technology	Defines the specific technology that will be used to build the system.

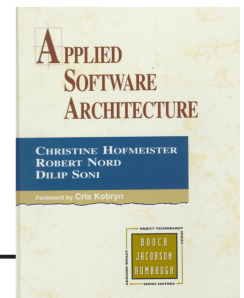




# Apéndice: Conjuntos de vistas

## Siemens

Christine Hofmeister, Robert Nord & Dilip Soni:  
**Applied Software Architecture**  
Addison-Wesley, 2000



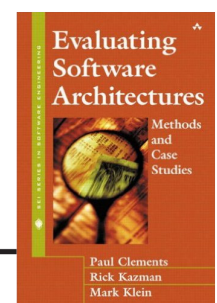
Viewpoint	Definition
Conceptual	The conceptual functional structure of the system that defines a set of conceptual components linked by a set of connectors.
Module	The concrete structure of the subsystems and modules that will be realized in the system, the interfaces exposed by the modules, the intermodule dependencies, and any layering constraints in the structure.
Execution	The runtime structure of the system in terms of processes, threads, interprocess communication elements, and so on along with a mapping of modules to runtime elements.
Code	The design-time layout of the system as source code and the intermediate and delivered binary elements created from it.



# Apéndice: Conjuntos de vistas

## SEI Viewtypes

Paul Clements, Rick Kazman & Mark Klein:  
**Evaluating Software Architectures.**  
Addison-Wesley, 2002.



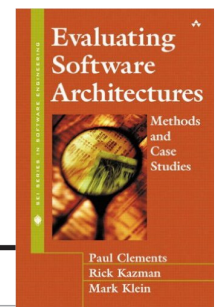
Viewtype	Definition
Component and Connector	<p>The Component and Connector viewtype is concerned with the system's runtime functional elements, their behaviors, and their interactions. The following styles defined for this viewtype all relate to commonly occurring runtime system organizations:</p> <ul style="list-style-type: none"><li>• <i>Pipe-and-Filter</i></li><li>• <i>Shared-Data</i></li><li>• <i>Publish-Subscribe</i></li><li>• <i>Client-Server</i></li><li>• <i>Peer-to-Peer</i></li><li>• <i>Communicating-Processes</i></li></ul>



# Apéndice: Conjuntos de vistas

## SEI Viewtypes

Paul Clements, Rick Kazman & Mark Klein:  
**Evaluating Software Architectures.**  
Addison-Wesley, 2002.



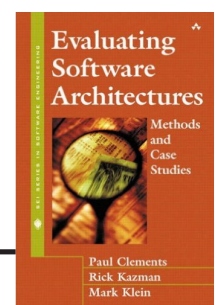
Viewtype	Definition
Module	<p>The Module viewtype is concerned with how the software comprising the system is structured as a set of implementation (code) units. The following styles are defined for the Module viewtype:</p> <ul style="list-style-type: none"><li>• <i>Uses</i>: for capturing intermodule usage dependencies</li><li>• <i>Generalization</i>: for capturing commonality and variation (inheritance) relationships between modules</li><li>• <i>Decomposition</i>: for specifying how modules are composed from simpler elements</li><li>• <i>Layered</i>: for specifying how modules are arranged in layers according to their level of abstraction</li></ul>



# Apéndice: Conjuntos de vistas

## SEI Viewtypes

Paul Clements, Rick Kazman & Mark Klein:  
**Evaluating Software Architectures.**  
Addison-Wesley, 2002.

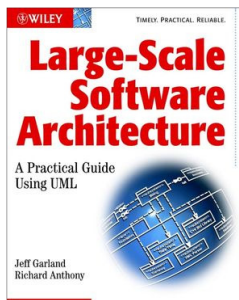


Viewtype	Definition
Allocation	<p>The Allocation viewtype is concerned with how relationships between the different parts of the system and different aspects of their environment are captured. The following styles are defined for this viewtype:</p> <ul style="list-style-type: none"><li>• <i>Deployment</i>: for specifying how software elements are mapped to elements of the deployment environment</li><li>• <i>Implementation</i>: for specifying how software modules are mapped to the development environment (such as their location in a codeline)</li><li>• <i>Work Assignment</i>: for mapping software modules to those responsible for creating, testing, and deploying them</li></ul>



# Apéndice: Conjuntos de vistas

Jeff Garland &  
Richard Anthony:  
**Large Scale  
Software  
Architecture.**  
Wiley, 2003.



Viewpoint	Definition
Analysis Focused	Illustrates how the elements of the system work together in response to a functional usage scenario
Analysis Interaction	Presents the interaction diagram used during problem analysis
Analysis Overall	Consolidates the contents of the Analysis Focused view into a single model
Component	Defines the system's architecturally significant components and their connections
Component Interaction	Illustrates how the components interact in order to make the system work
Component State	Presents the state model(s) for a component or set of closely related components
Context	Defines the context within which the system exists, in terms of external actors and their interactions with the system
Deployment	Shows how software components are mapped to hardware entities in order to be executed
Layered Subsystem	Illustrates the subsystems to be implemented and the layers in the software design structure
Logical Data	Presents the logical view of the architecturally significant data structure
Physical Data	Presents the physical view of the architecturally significant data structure
Process	Defines the runtime concurrency structure (operating system processes that the system's components will be packaged into and interprocess communication mechanisms that will allow communication between them)
Process State	Presents the state transition model for the system's processes
Subsystem Interface Dependency	Defines the dependencies that exist between subsystems and the interfaces of other subsystems

