DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

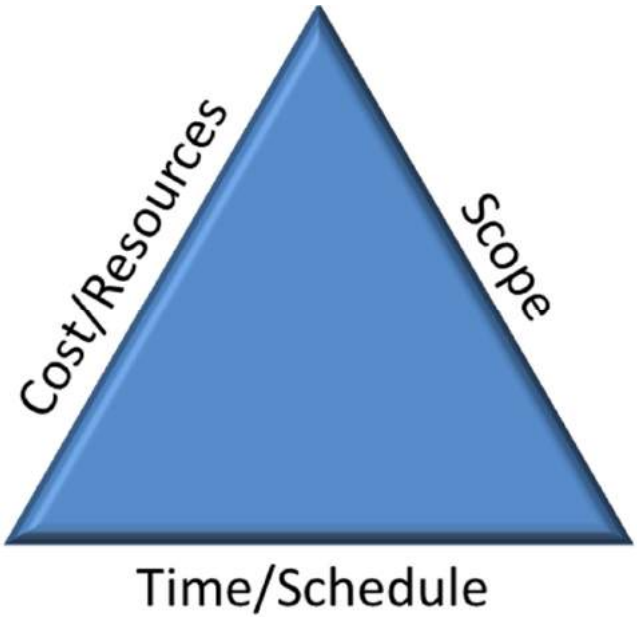
Arquitecturas software

© Fernando Berzal, 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.

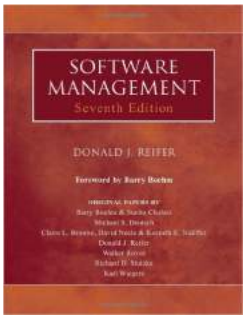
Introducción

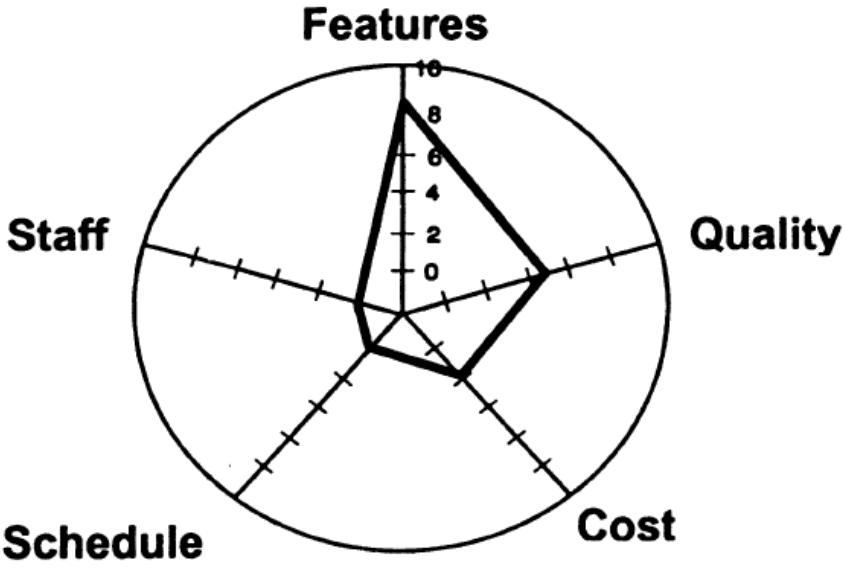


"I can make it for you fast, cheap, or good. Pick any two."
https://en.wikipedia.org/wiki/Project_management_triangle

Introducción

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

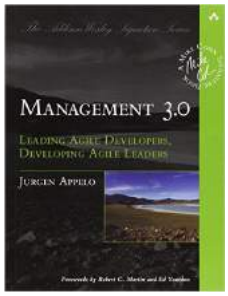


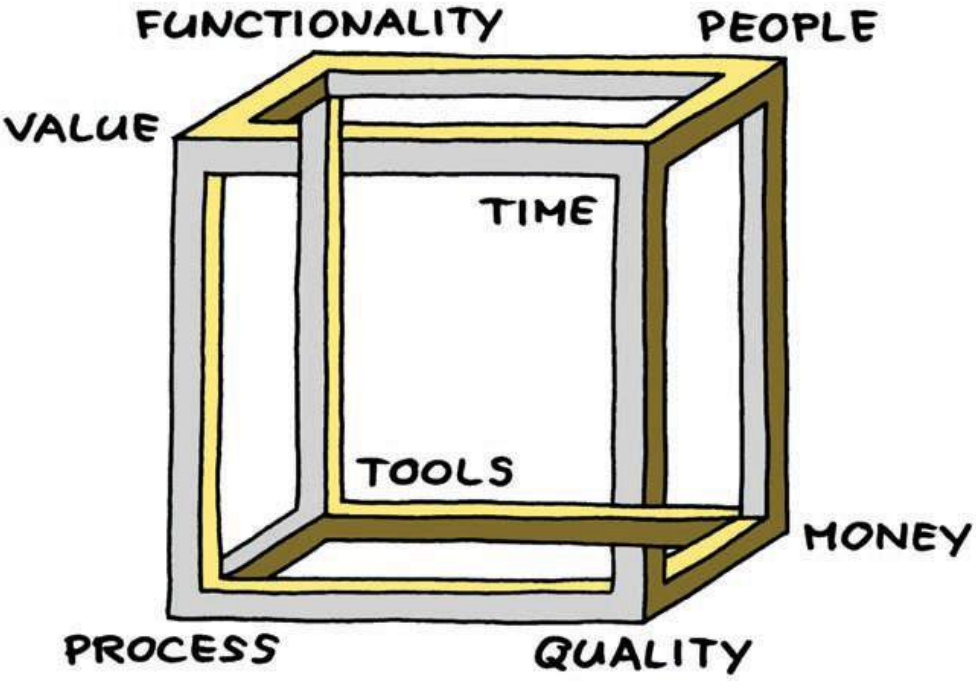


Karl E. Wiegers: "Creating a Software Engineering Culture", 1996

Introducción

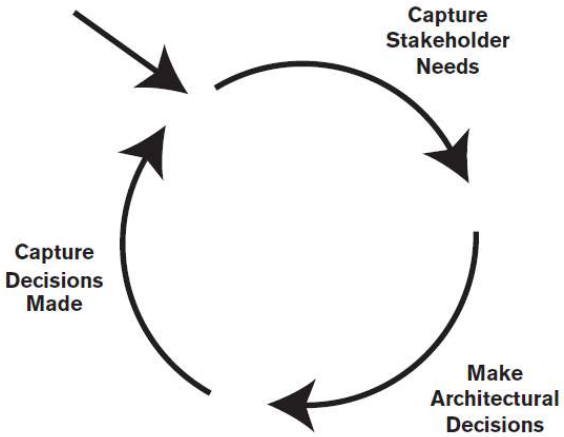
Puede ser difícil equilibrar todas las dimensiones clave…

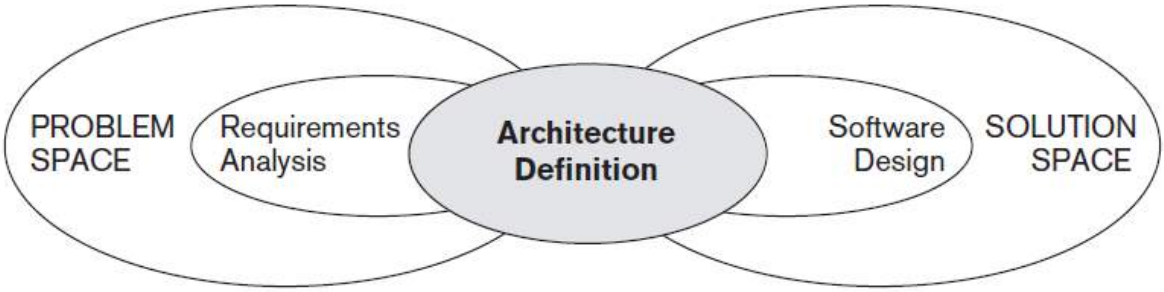




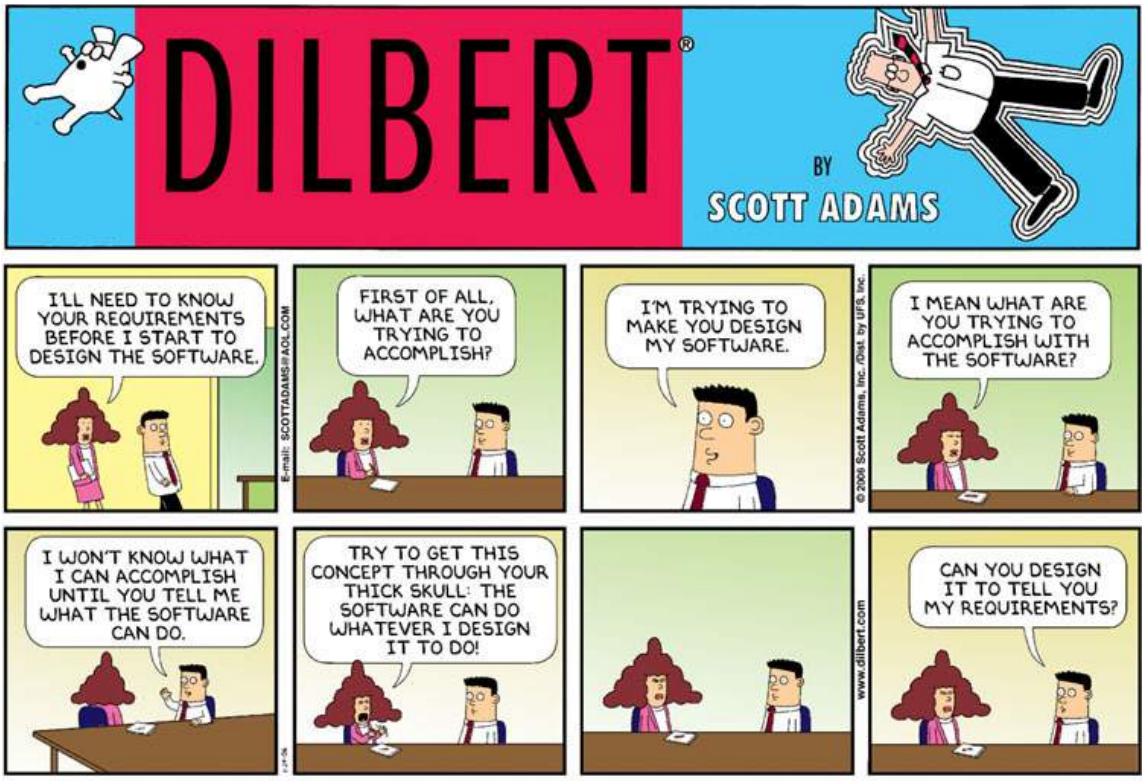
Introducción

El rol del arquitecto



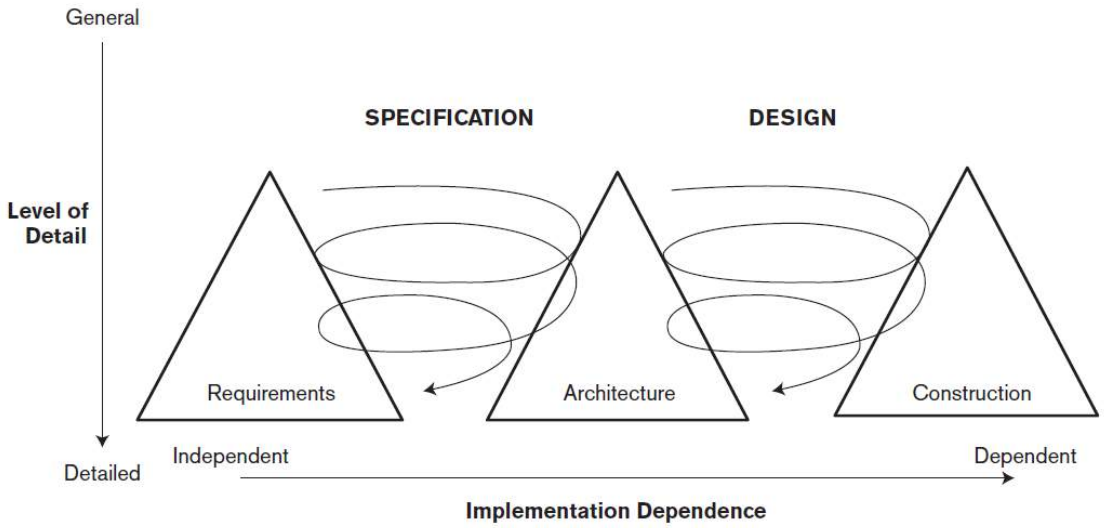


Introducción



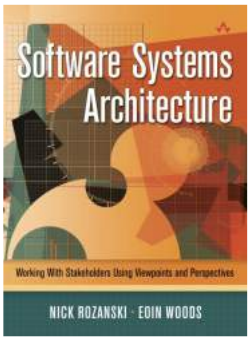
Introducción

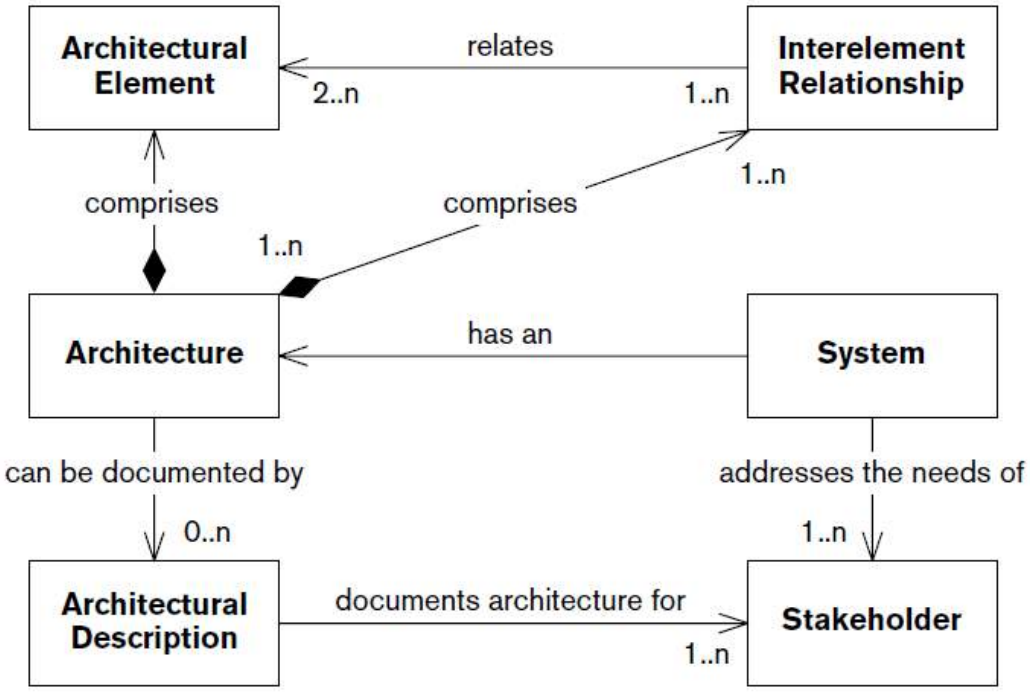
El modelo de 3 picos



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

Introducción





Vistas

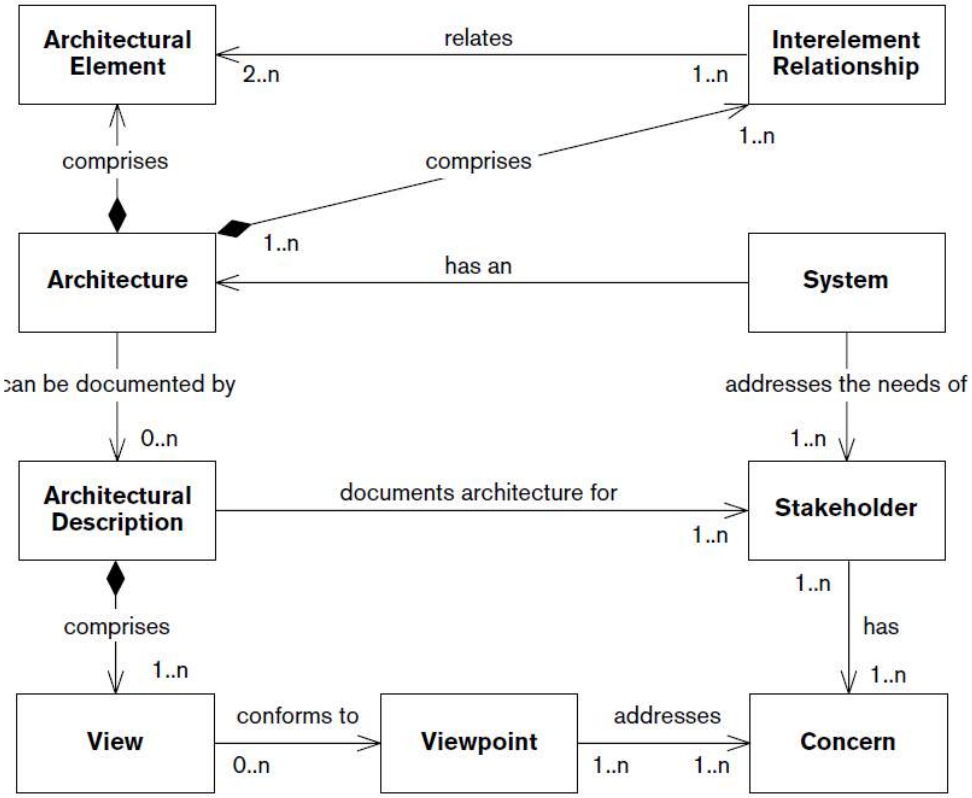
OBSERVACIÓN

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

IDEA

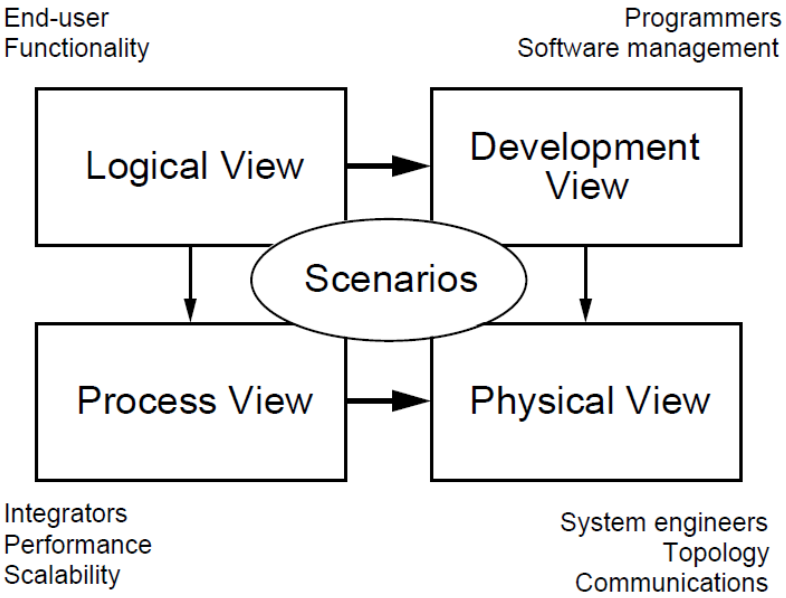
Un sistema complejo se describe mucho mejor mediante un conjunto de vistas relacionadas entre sí, que de forma colectiva ilustran sus características y demuestran que satisface sus objetivos.

Vistas



Vistas

Modelo 4+1

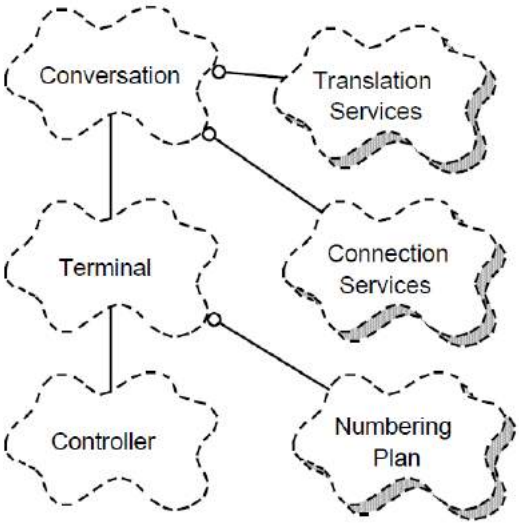


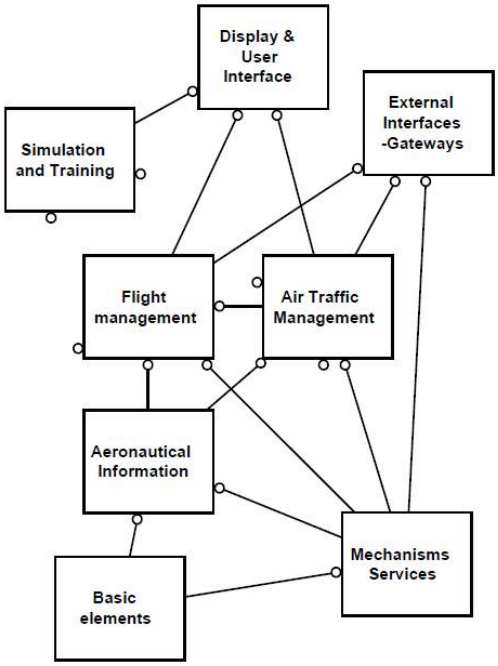
Phillipe Kruchten (Rational Corporation):
Architectural Blueprints:
The 4+1 View Model of Software Architecture
IEEE Software 12(6):42-50, November/December 1995

Vistas

Modelo 4+1: Vista lógica

Modelo OO del diseño





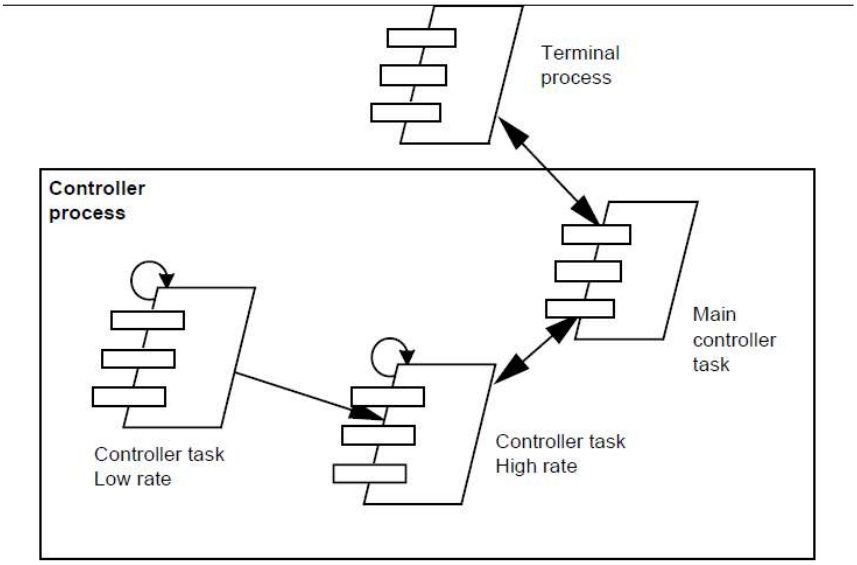
PABX

Control de tráfico aéreo

Vistas

Modelo 4+1: Vista de procesos

Concurrencia & sincronización

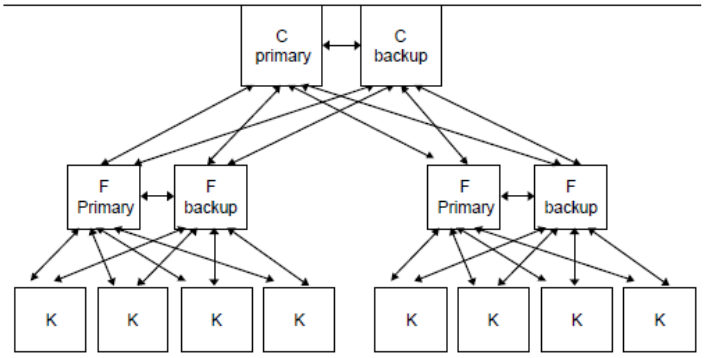


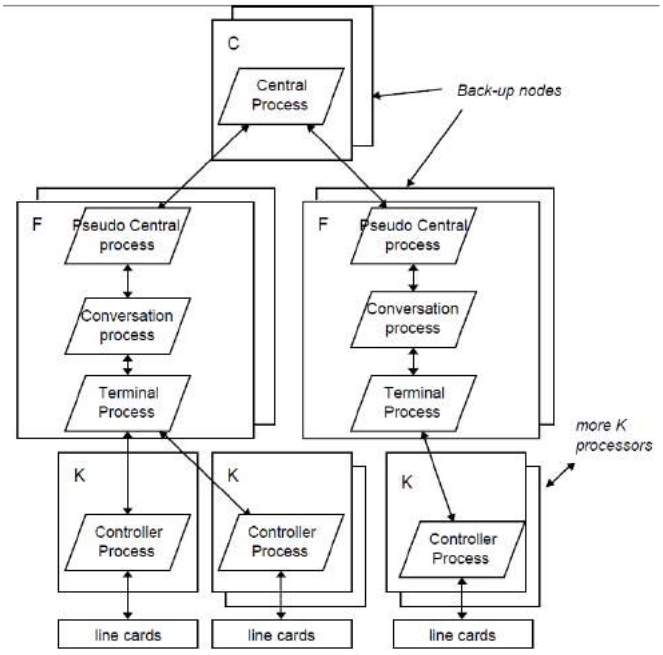
PABX

Vistas

Modelo 4+1: Vista física

Hardware & distribución



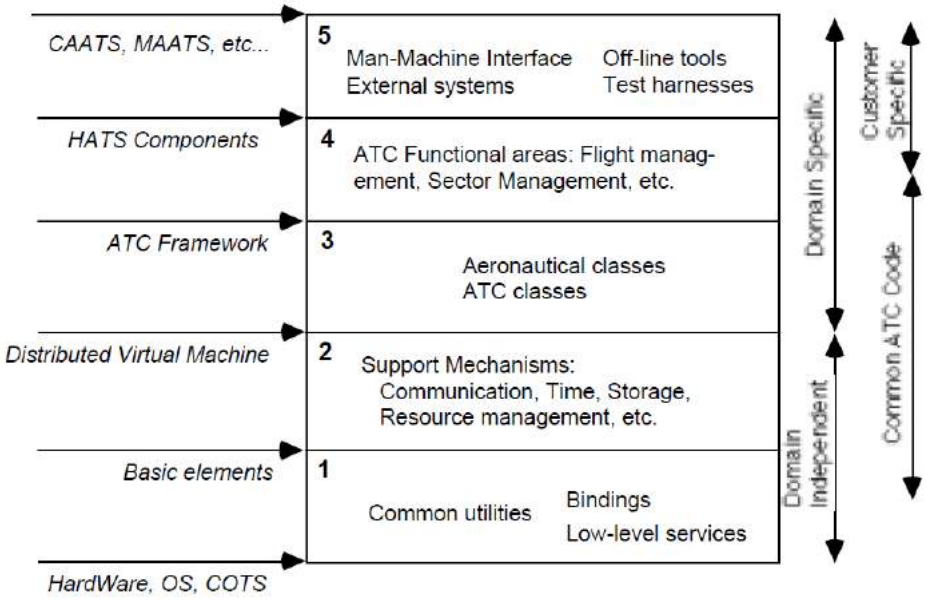


PABX

Vistas

Modelo 4+1: Vista de desarrollo

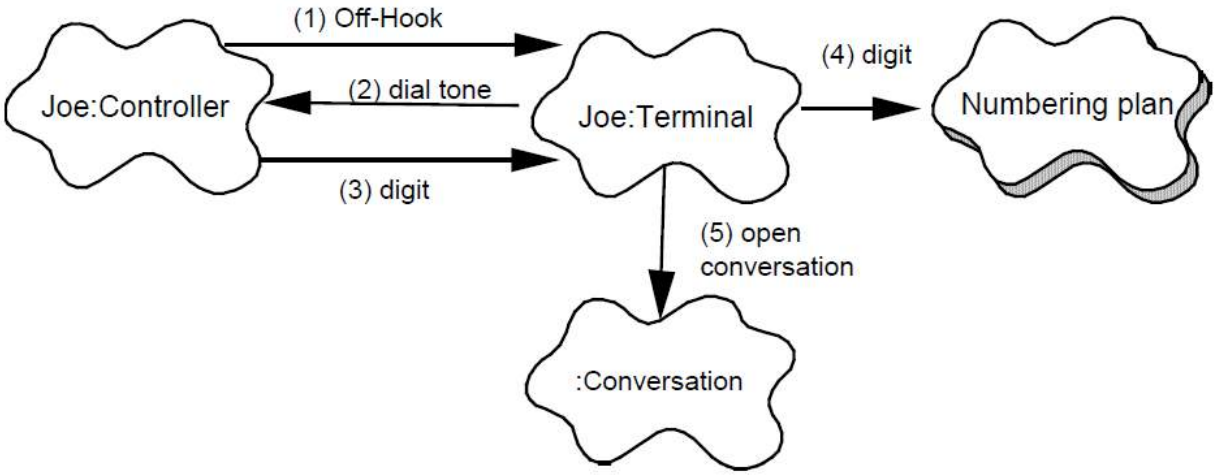
Organización de los módulos



Capas en un sistema de control de tráfico aéreo

Modelo 4+1: Escenarios

Casos de uso seleccionados



PABX

Modelo 4+1

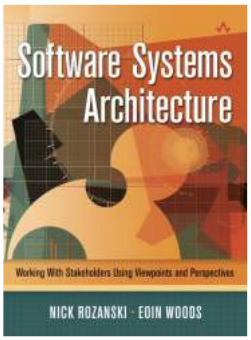
Resumen

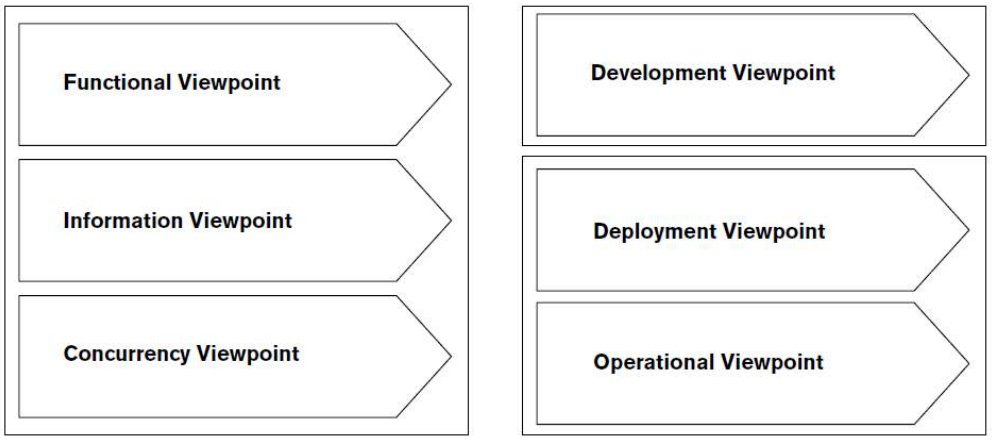
View	*Logical*	*Process*	*Development*	*Physical*	*Scenarios*
Components	Class	Task	Module, Subsystem	Node	Step, Scripts
Connectors	association, inheritance, containment	Rendez-vous, Message, broadcast, RPC, etc.	compilation dependency, "with" clause, "include"	Communica-tion medium, LAN, WAN, bus, etc.	
Containers	Class category	Process	Subsystem (library)	Physical subsystem	Web

Stakeholders	End-user	System designer, integrator	Developer, manager	System designer	End-user, developer
Concerns	Functionality	Performance, availability, S/W fault-tolerance, integrity	Organization, reuse, portability, line-of-product	Scalability, performance,av ailability	Understand-ability
Tool support	Rose	UNAS/SALE DADS	Apex, SoDA	UNAS, Openview DADS	Rose

Vistas

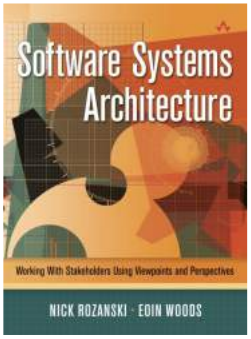
Modelo alternativo
Puntos de vista…





Vistas

Modelo alternativo
Punto de vista: Funcional



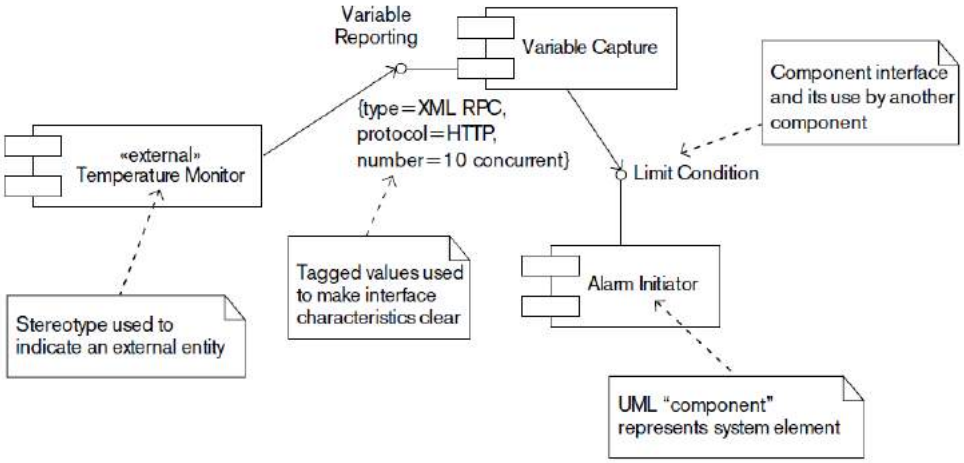
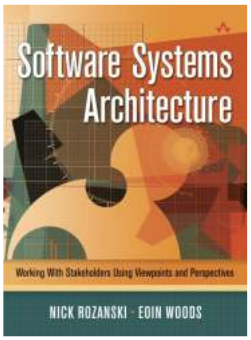


Diagrama de componentes UML

Vistas

Modelo alternativo

Punto de vista: Información



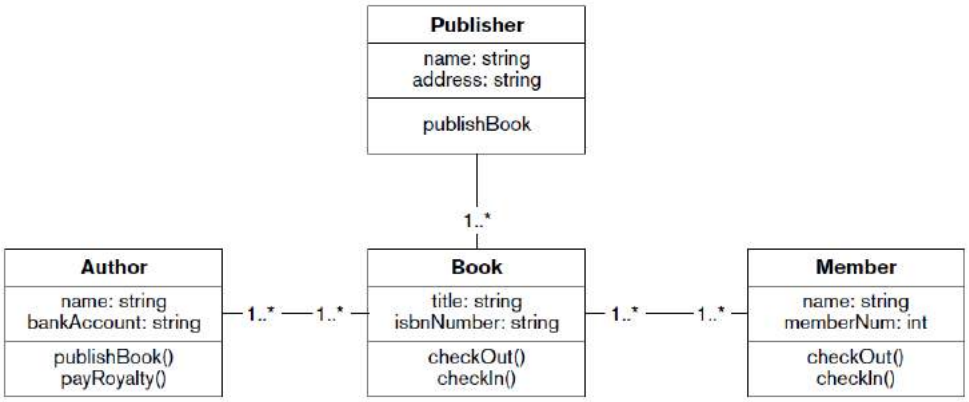
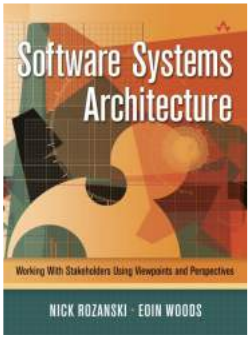


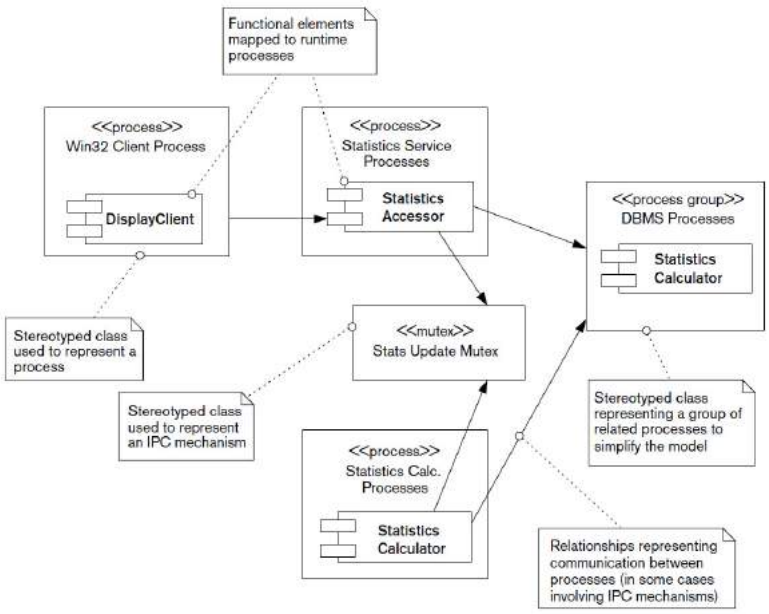
Diagrama de clases UML

Vistas

Modelo alternativo

Punto de vista: Concurrencia



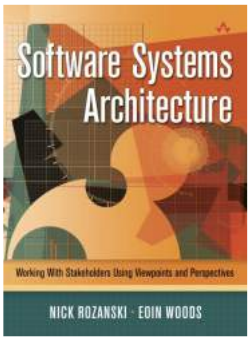


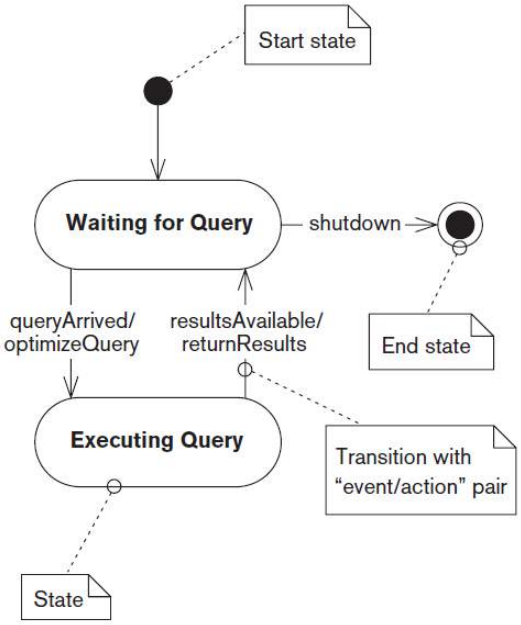
Uso de estereotipos en UML

Vistas

Modelo alternativo

Punto de vista: Concurrencia



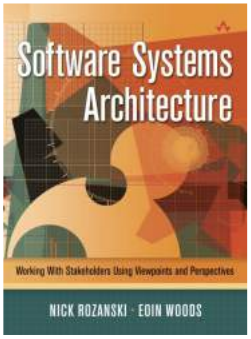


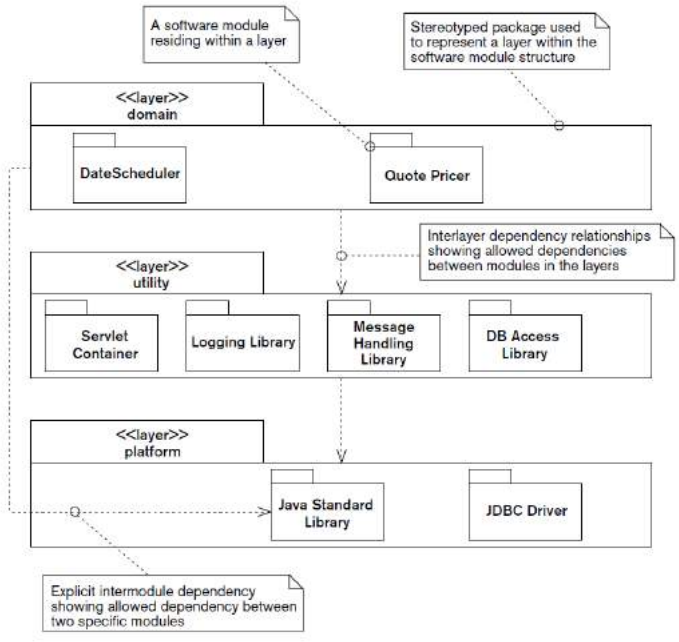
Diagramas de estados en UML

Vistas

Modelo alternativo

Punto de vista: Desarrollo



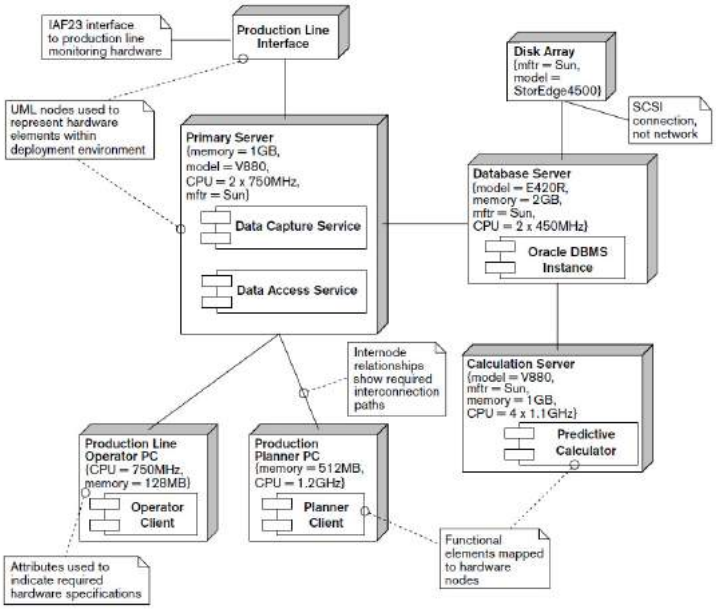


Diagramas de componentes en UML: Paquetes

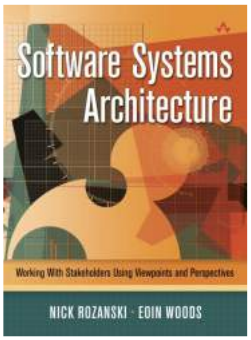
Vistas

Modelo alternativo

Punto de vista: Despliegue



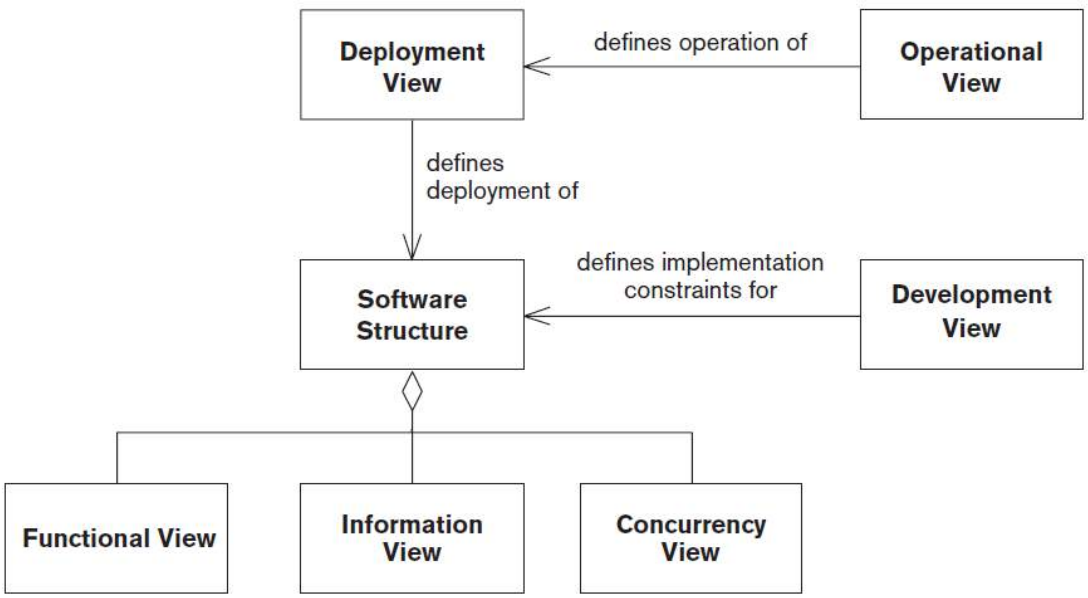
Diagramas de despliegue en UML

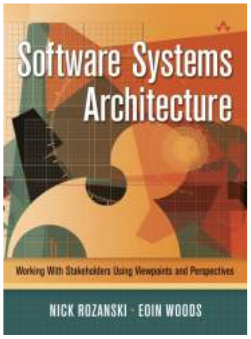


Vistas

Modelo alternativo

Puntos de vista...

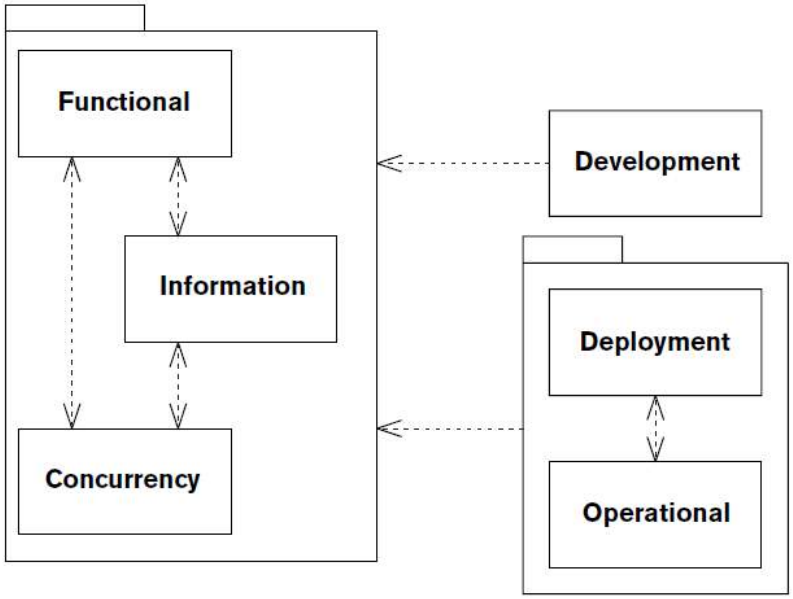


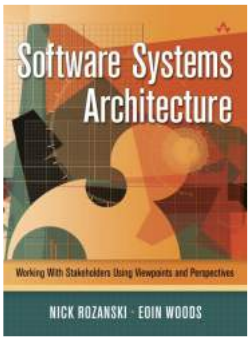


Vistas

Modelo alternativo

Puntos de vista…



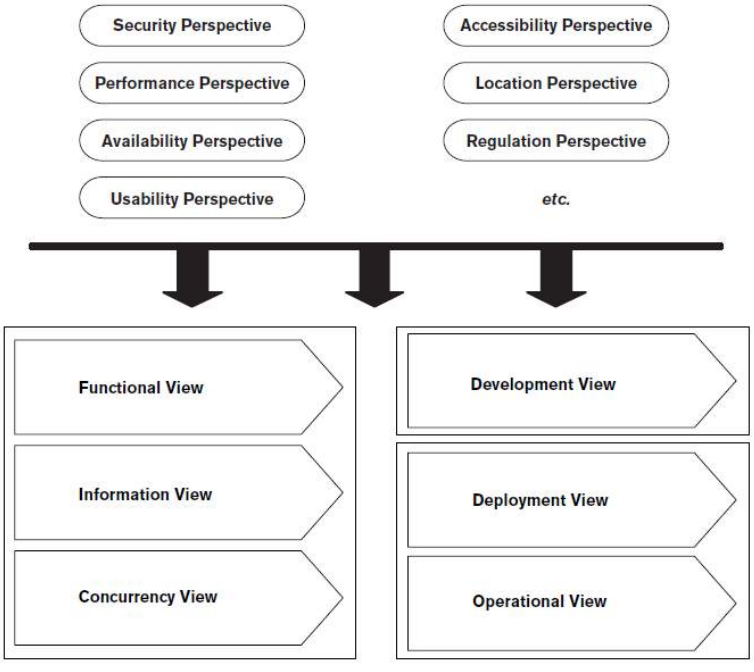


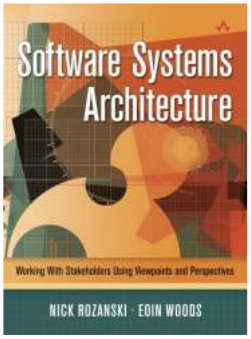
Dependencias entre puntos de vista

Vistas

Modelo alternativo

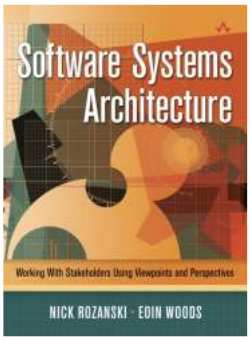
… y perspectivas

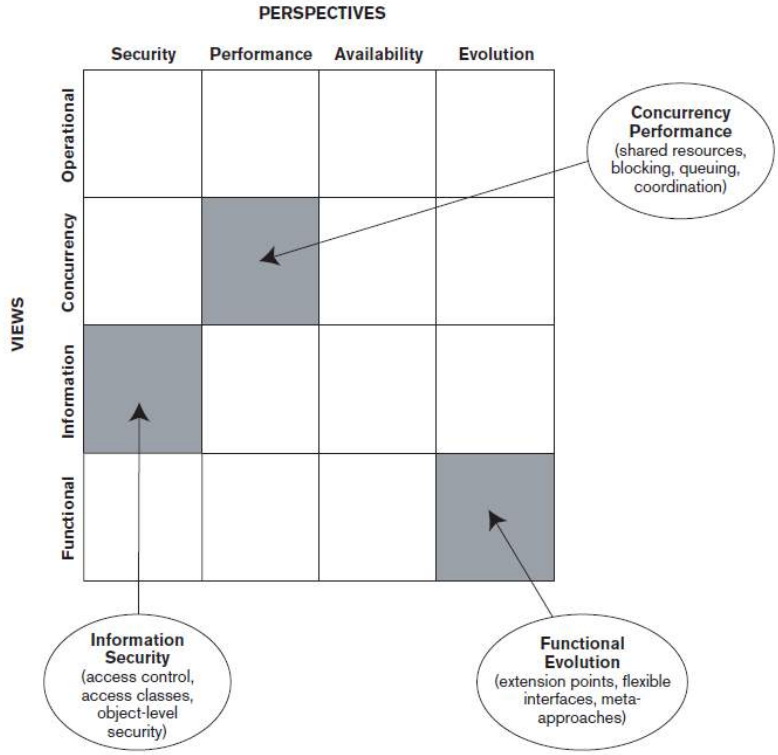




Vistas

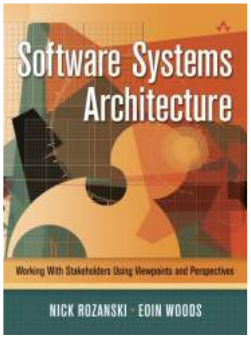
Modelo alternativo

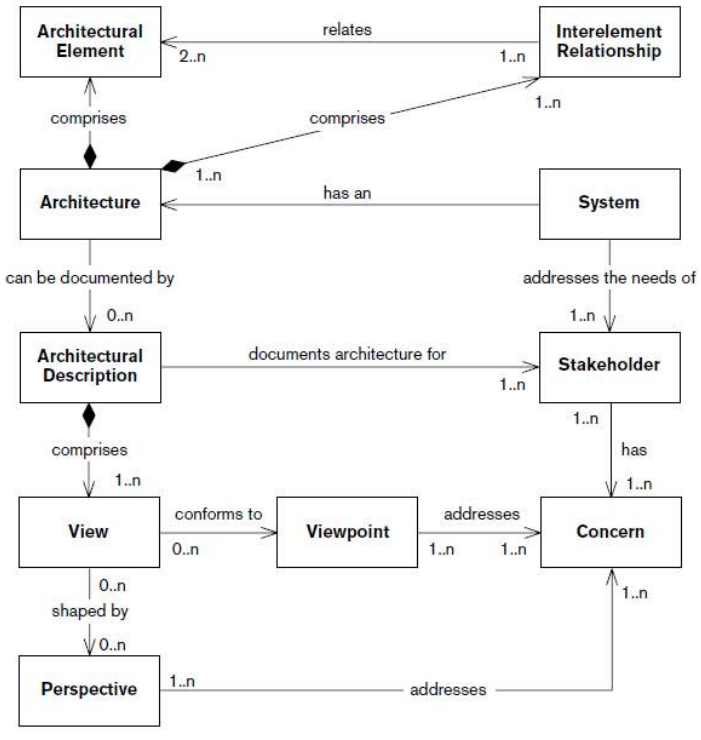




Vistas

Modelo alternativo





Vistas

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.

Vistas

Limitaciones

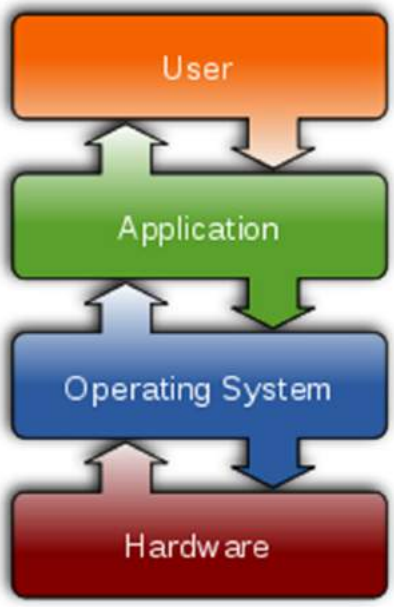
- Inconsistencias entre las distintas vistas.
- Selección de un conjunto erróneo de vistas.
- Fragmentación de la descripción arquitectónica.

Patrones arquitectónicos

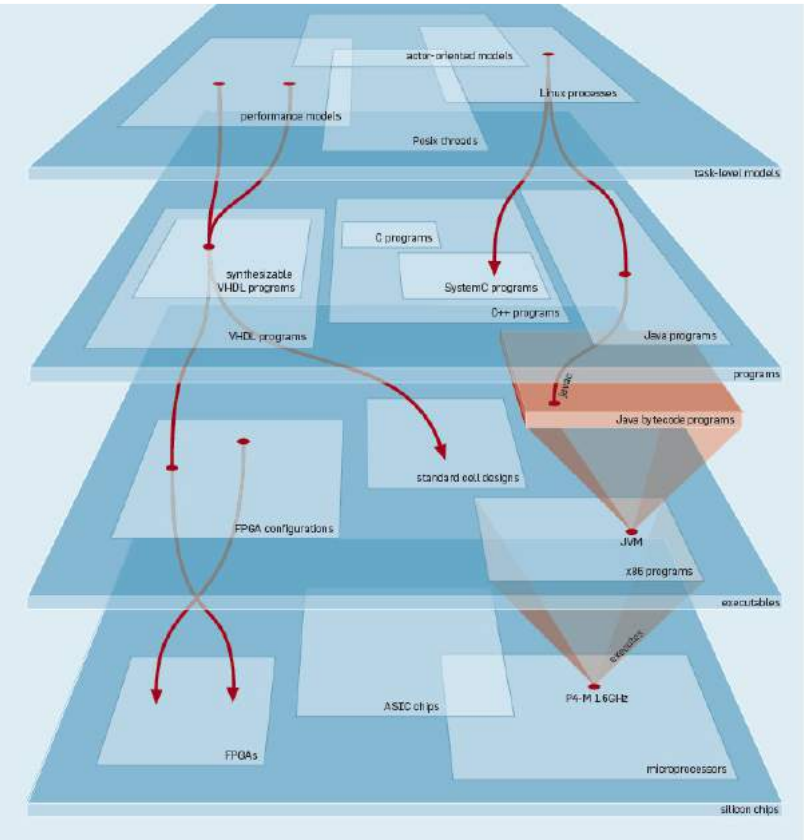
- Arquitecturas basadas en capas.
- Arquitecturas basadas en flujos de datos.
- Arquitecturas con pizarra

Patrones arquitectónicos
Capas

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



Patrones arquitectónicos
Capas



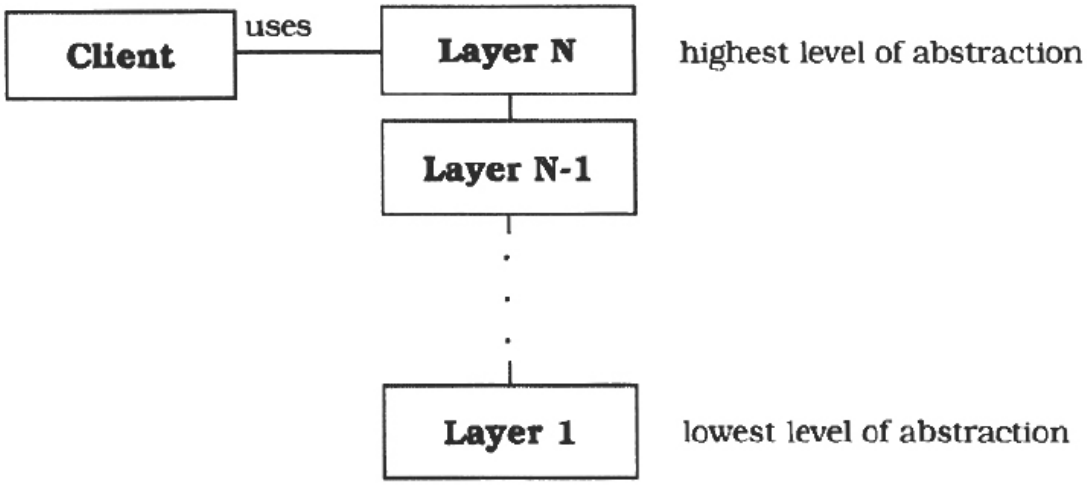
Capas de abstracción en computación

Patrones arquitectónicos
Capas

Class Layer J	***Collaborator*** • Layer J-1
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J-1.	

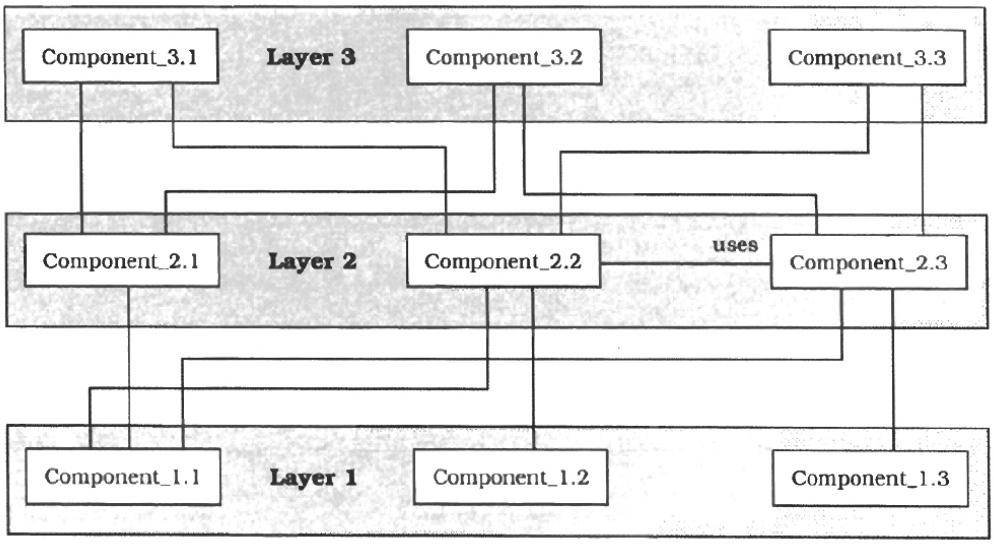
Tarjeta CRC de una capa [POSA]

Patrones arquitectónicos
Capas



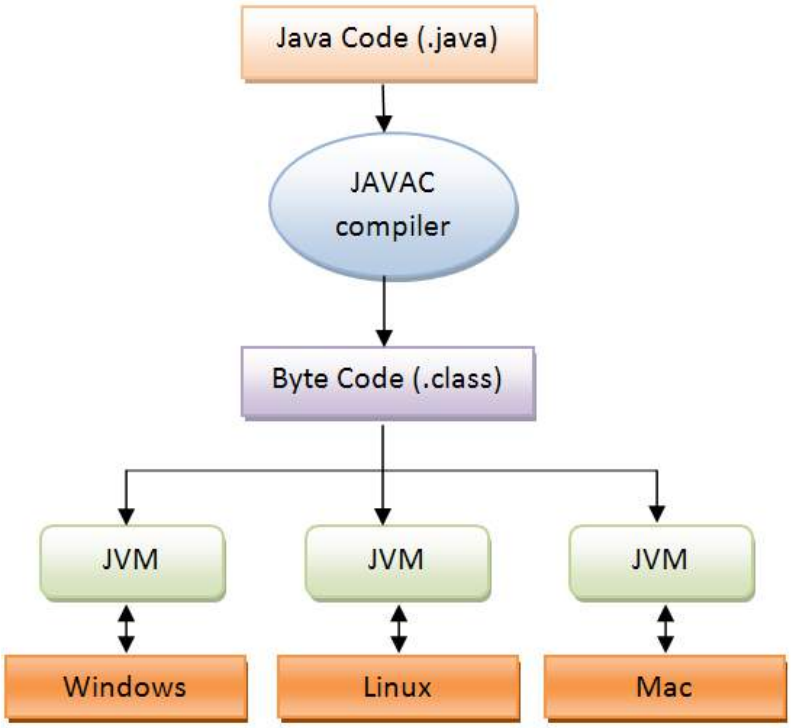
Estructura [POSA]

Patrones arquitectónicos
Capas



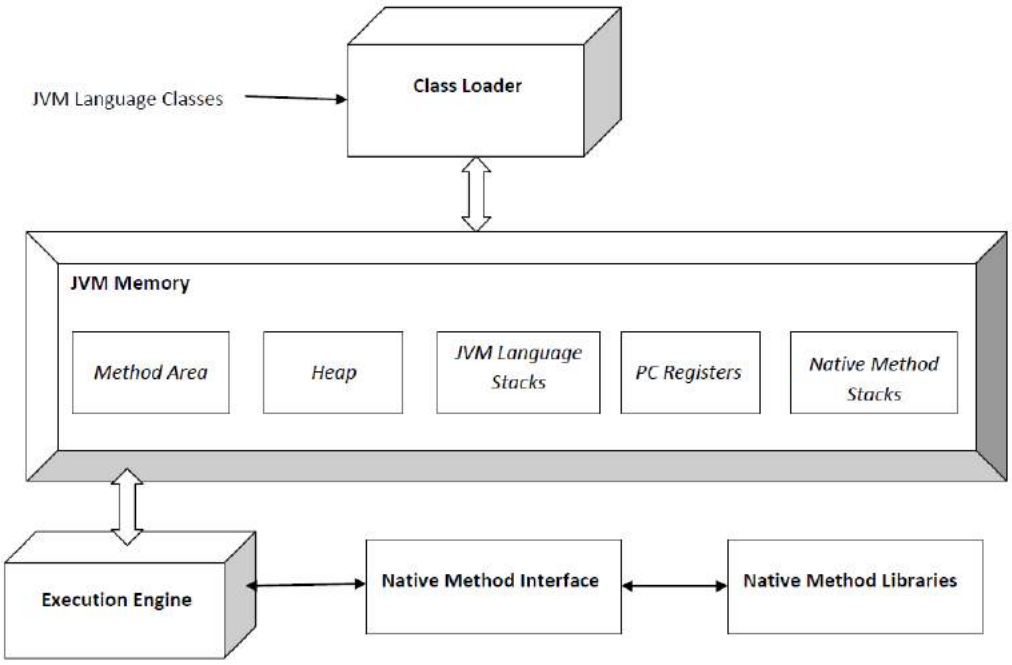
Estructura [POSA]

La máquina virtual Java (JVM)

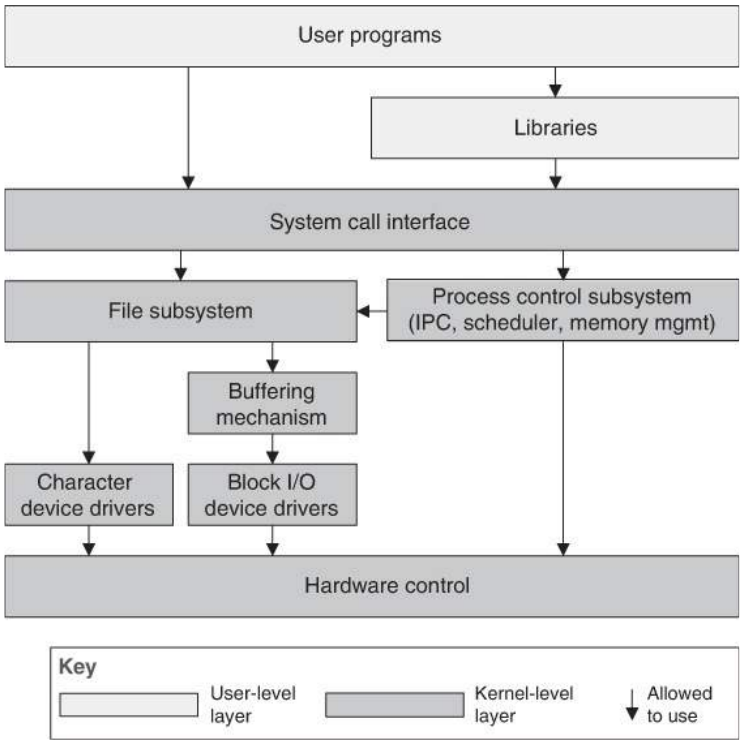


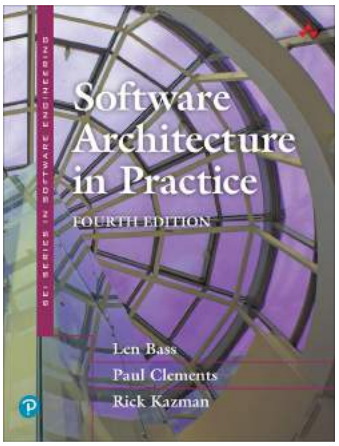
Patrones arquitectónicos
Capas

La máquina virtual Java (JVM)



Sistemas operativos

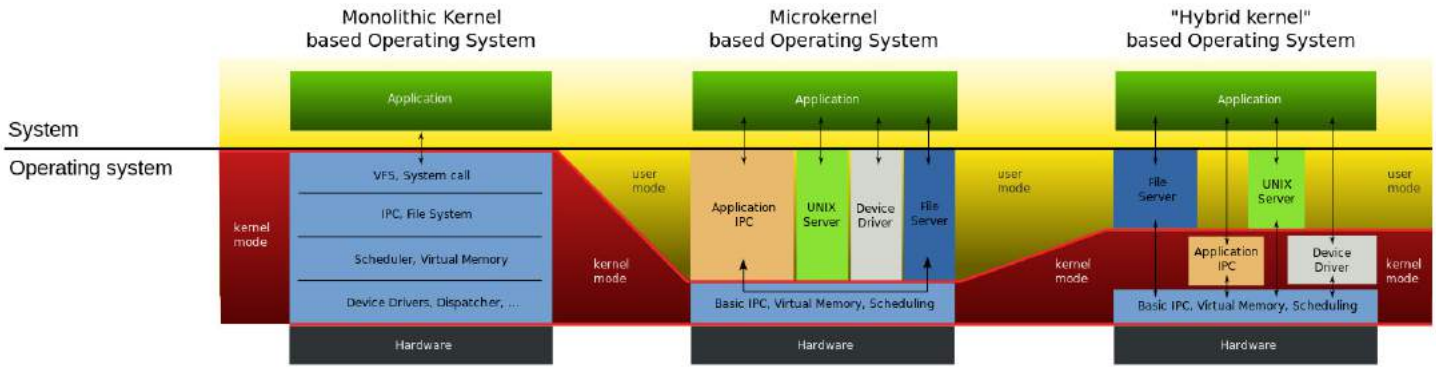




Patrones arquitectónicos
Capas

Sistemas operativos

Kernel monolítico vs. Microkernel



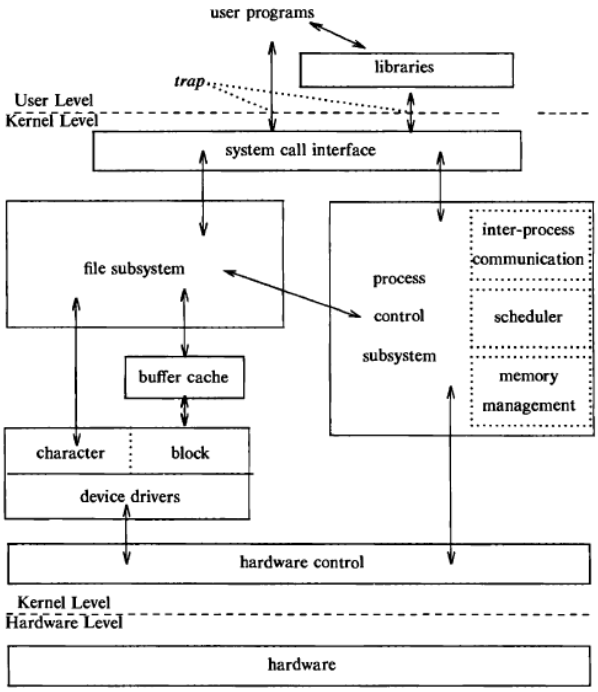
Monolítico: BSD, AIX, Linux, Multics, Windows 9x…

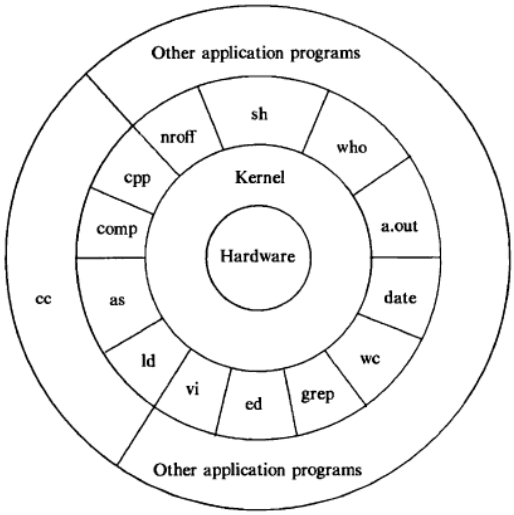
Microkernel: Mach, RTOS (QNX, ChorusOS, Integrity)…

Híbrido: Windows NT…

https://en.wikipedia.org/wiki/Kernel_(operating_system)

Sistemas operativos: UNIX

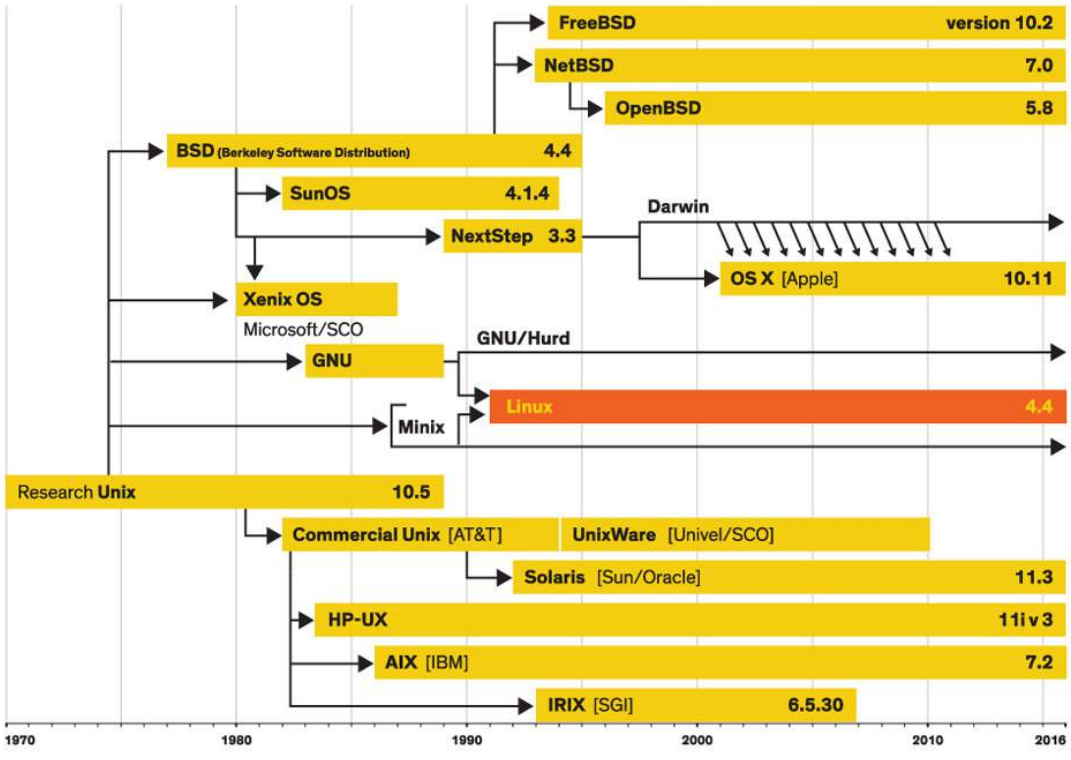




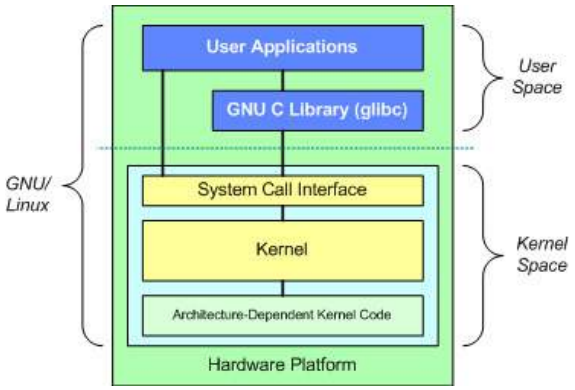
Bach: The Design of the UNIX Operating System, 1986

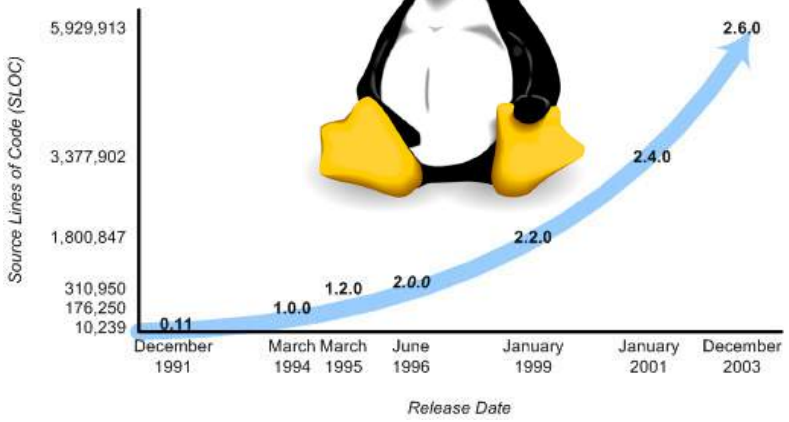
Patrones arquitectónicos
Capas

Sistemas operativos: UNIX



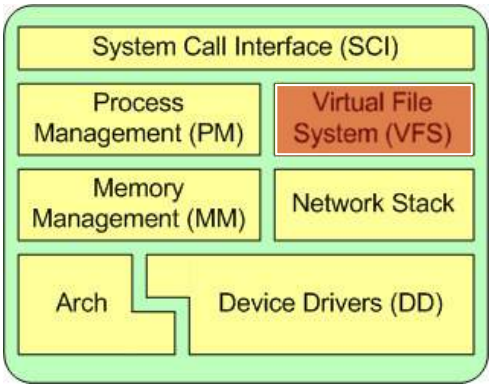
Sistemas operativos: Linux

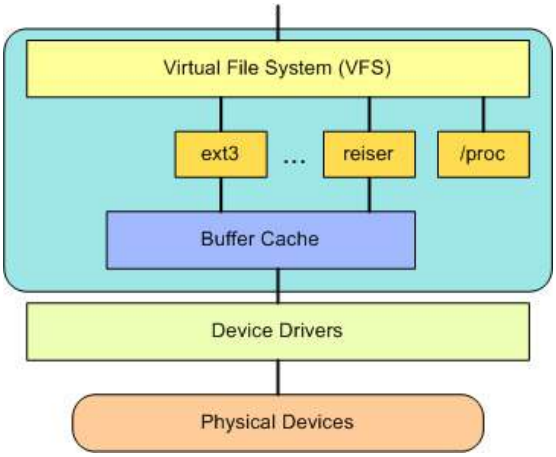




Versión 4.3 (2015)
>20MLOC

Sistemas operativos: Linux

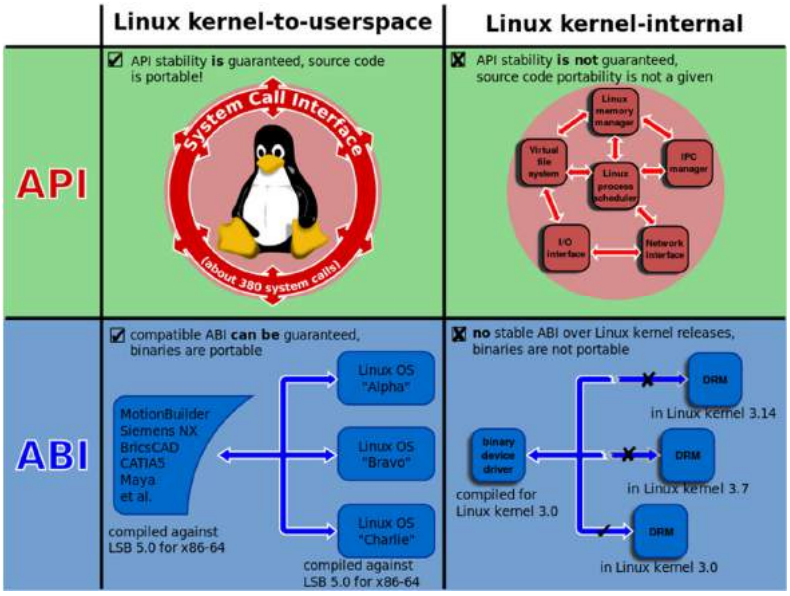




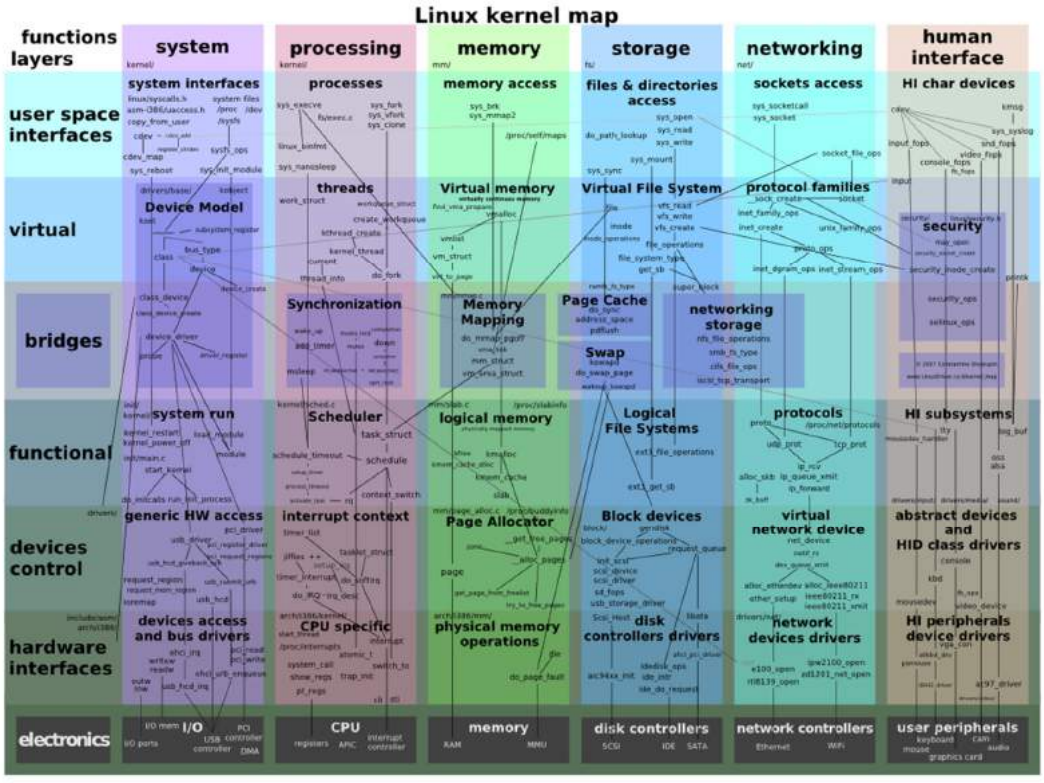
Sistemas operativos: Linux

API: Application Programming Interface

ABI: Application Binary Interface

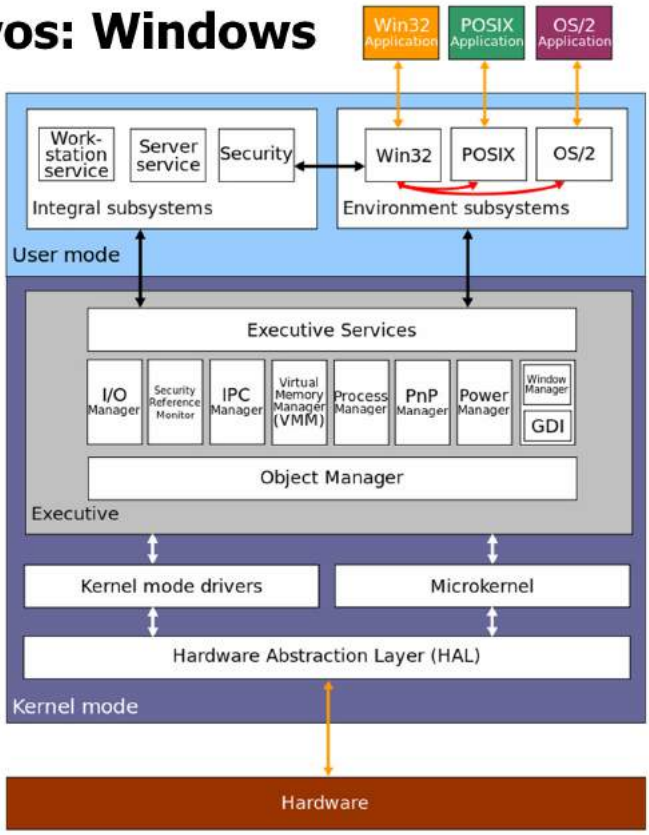


Sistemas operativos: Linux

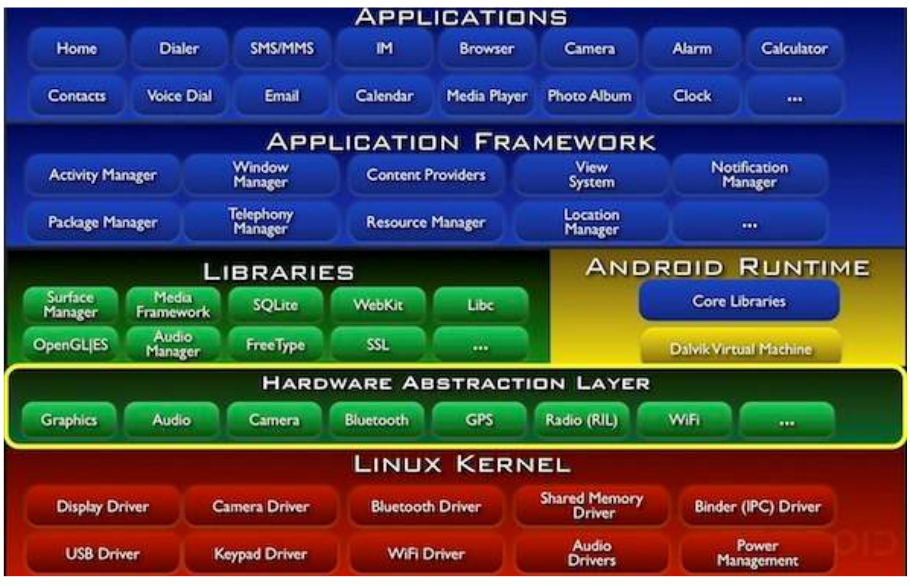


Sistemas operativos: Windows

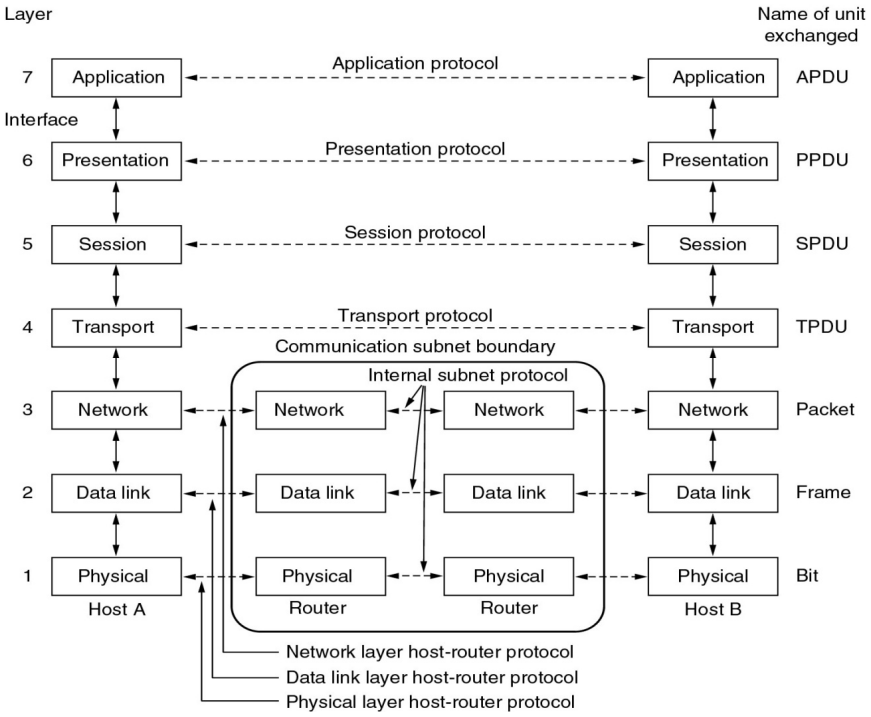
Windows NT
(1989…)



Sistemas operativos: Android

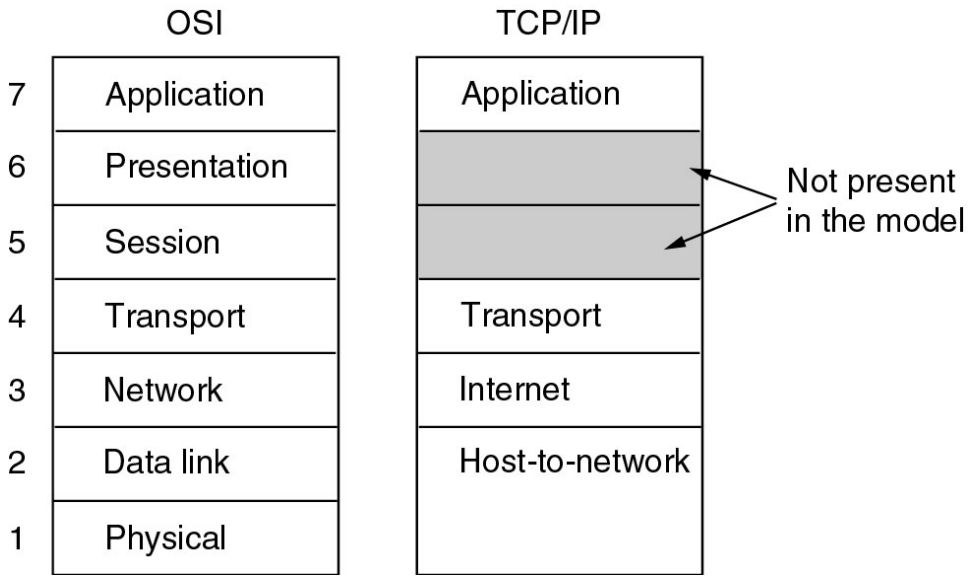


Comunicaciones: El modelo de referencia OSI

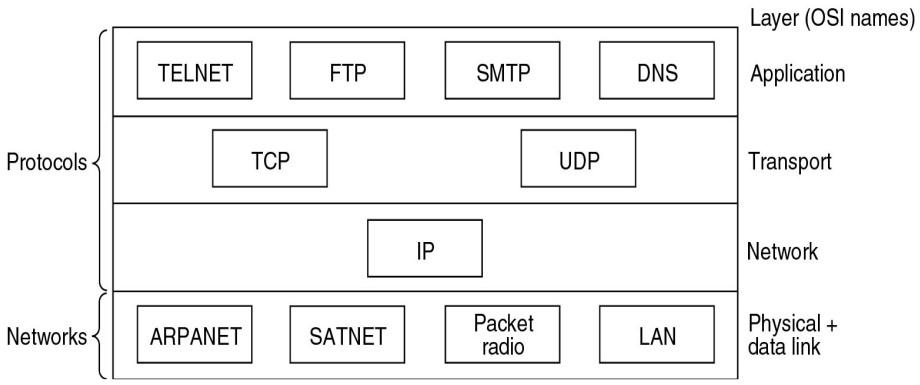


Patrones arquitectónicos
Capas

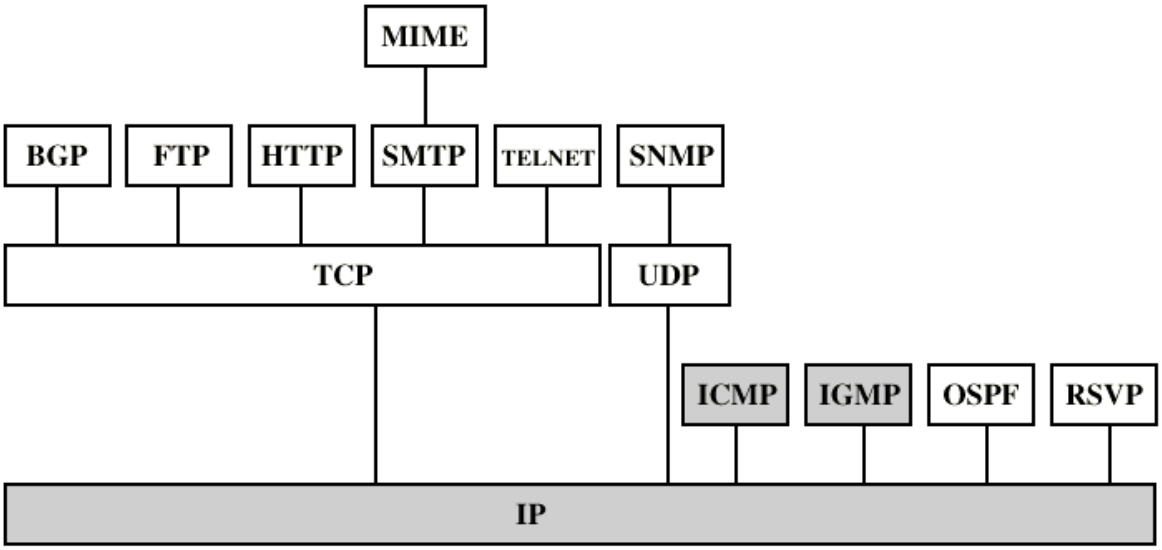
Comunicaciones: El modelo TCP/IP



Comunicaciones: El modelo TCP/IP

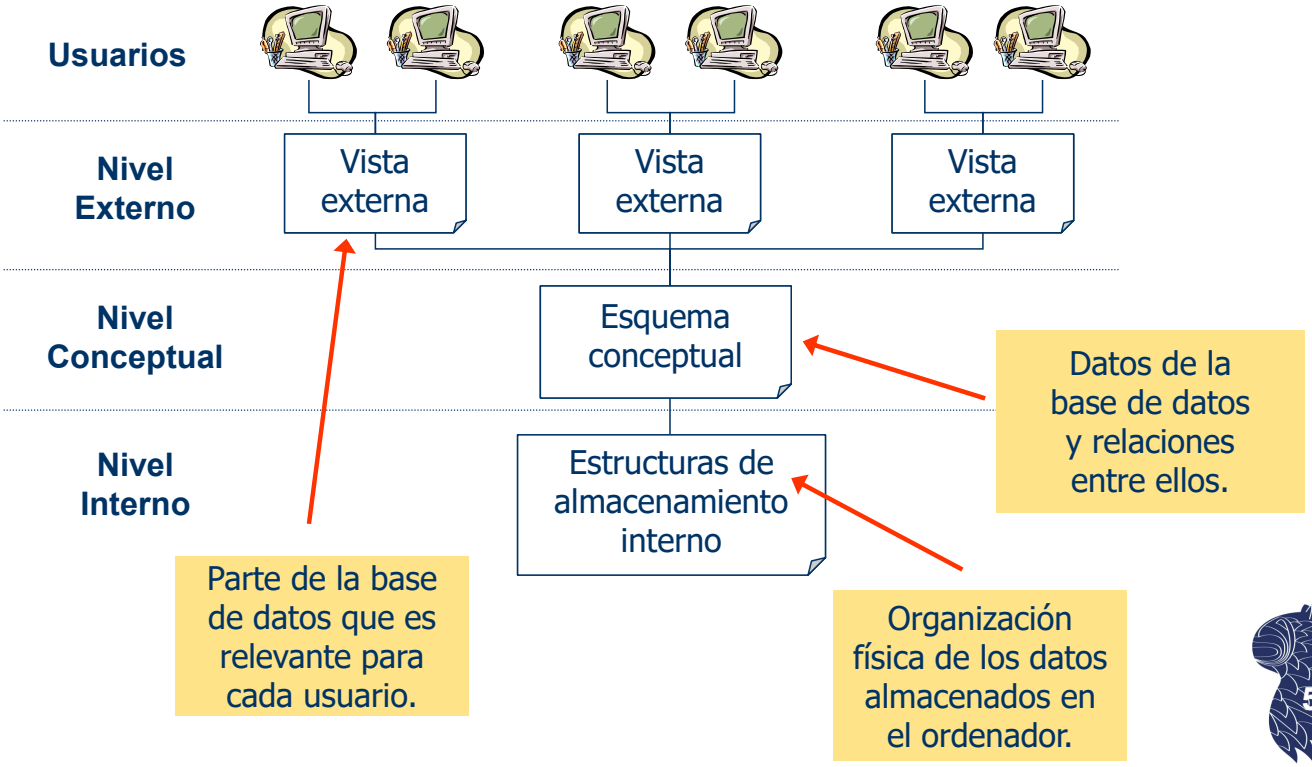


Comunicaciones: El modelo TCP/IP

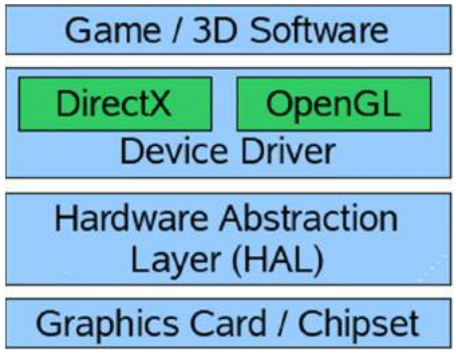


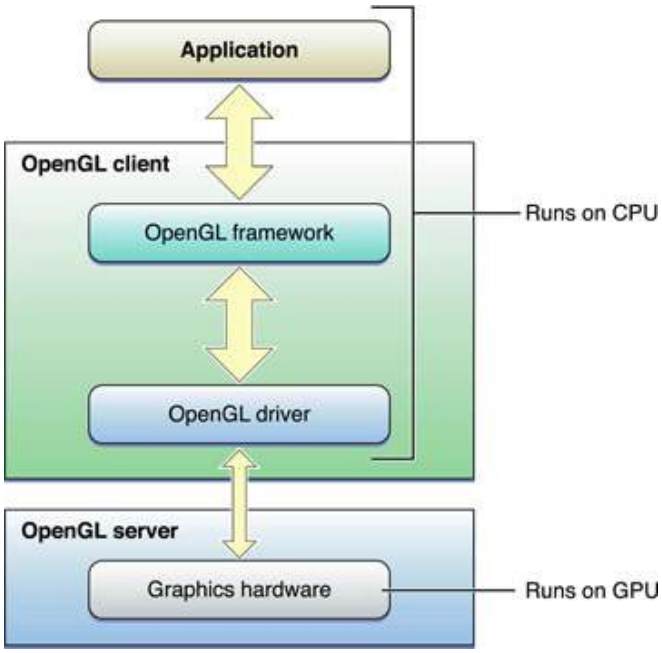
Familia de protocolos TCP/IP

Arquitectura ANSI/SPARC de un DBMS



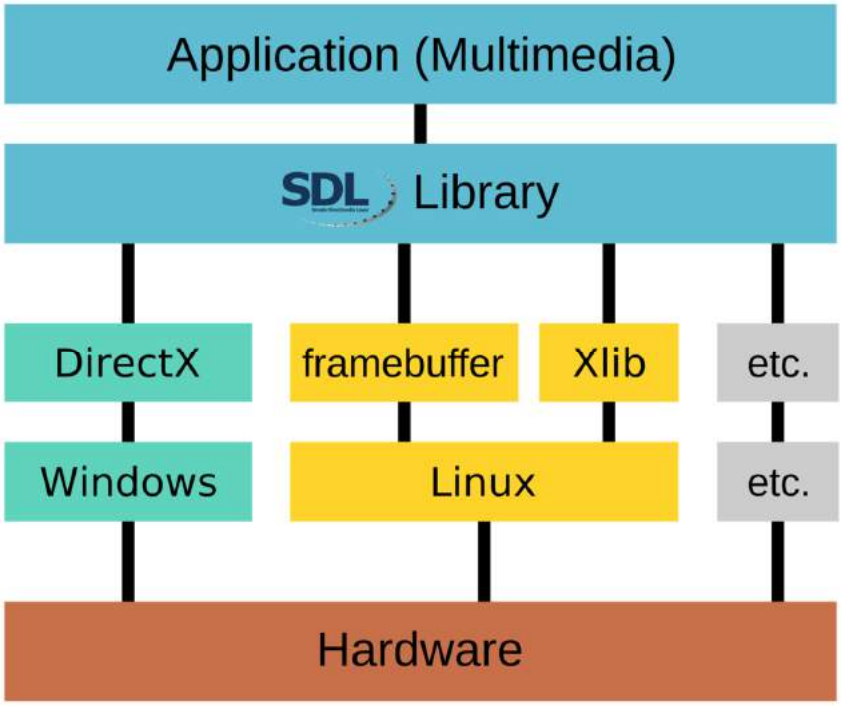
OpenGL/DirectX





Patrones arquitectónicos
Capas

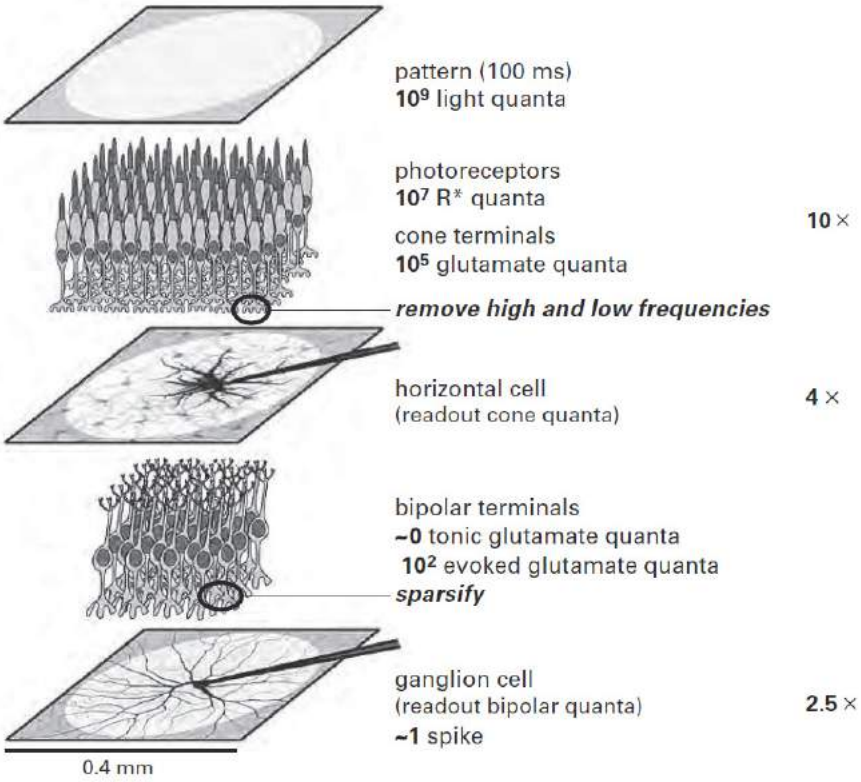
Motores de videojuegos



Patrones arquitectónicos
Capas

No sólo en software…

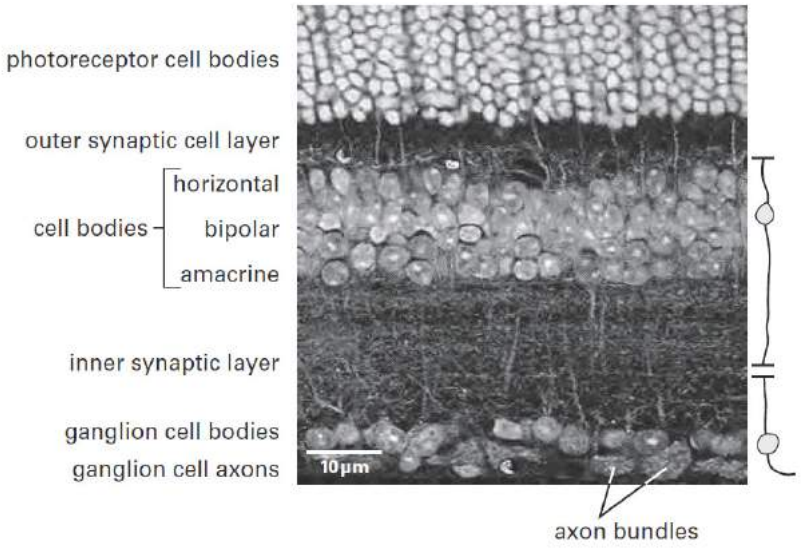
Retina

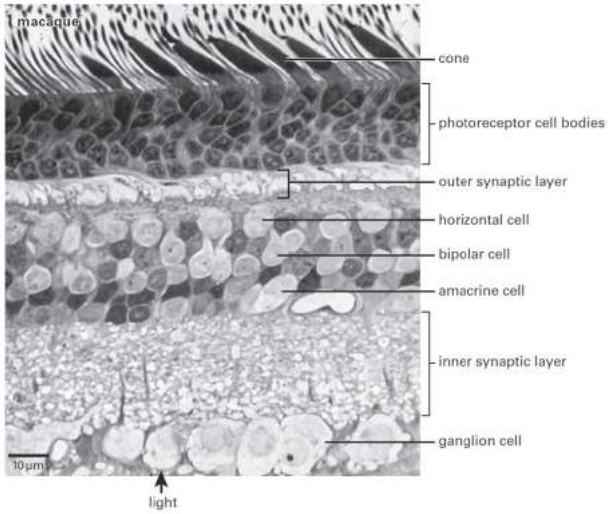


Patrones arquitectónicos
Capas

No sólo en software…

Retina

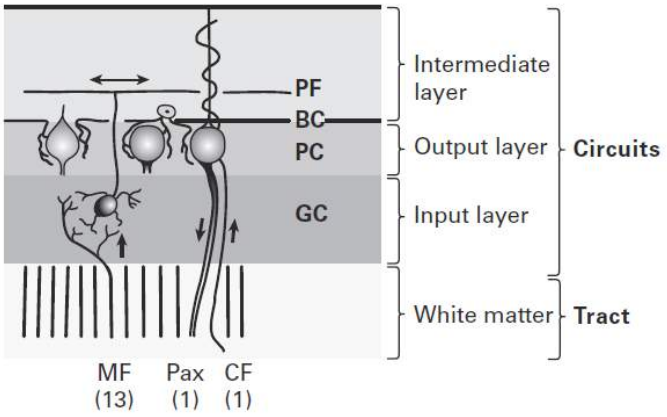


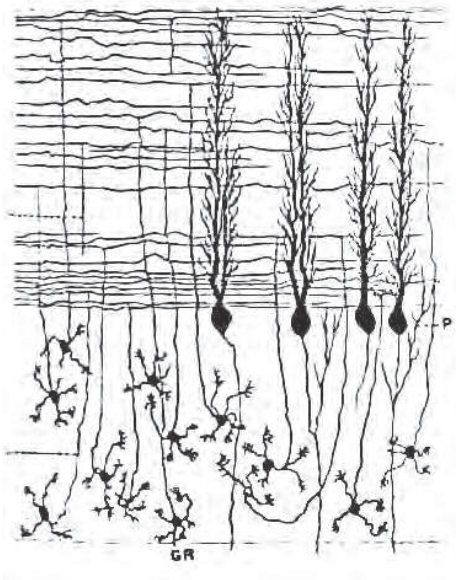


Patrones arquitectónicos
Capas

No sólo en software…

Córtex cerebelar

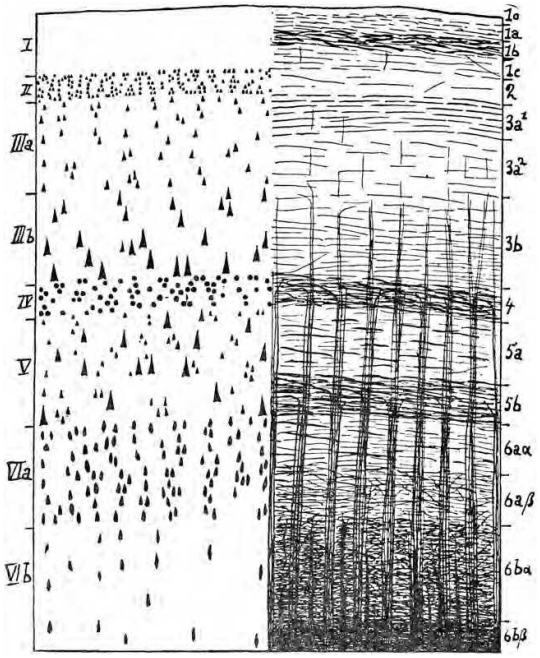


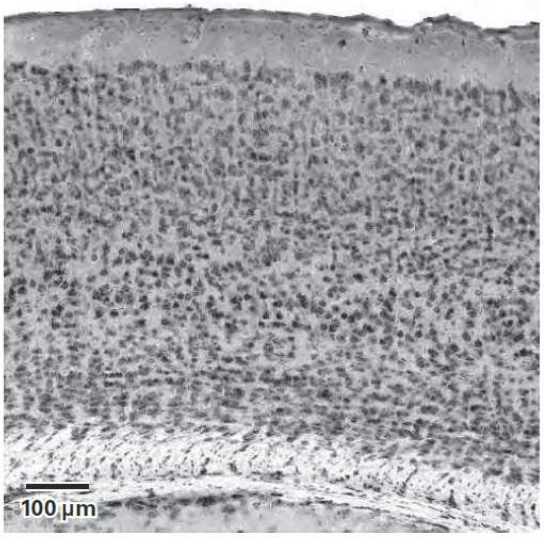


Patrones arquitectónicos
Capas

No sólo en software…

Córtex cerebral

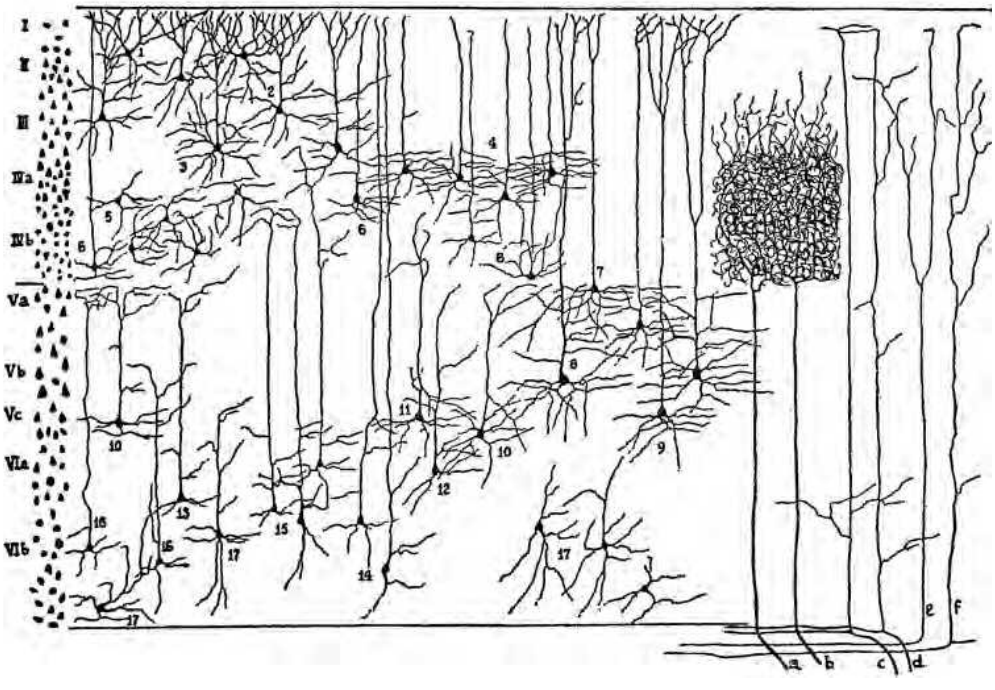




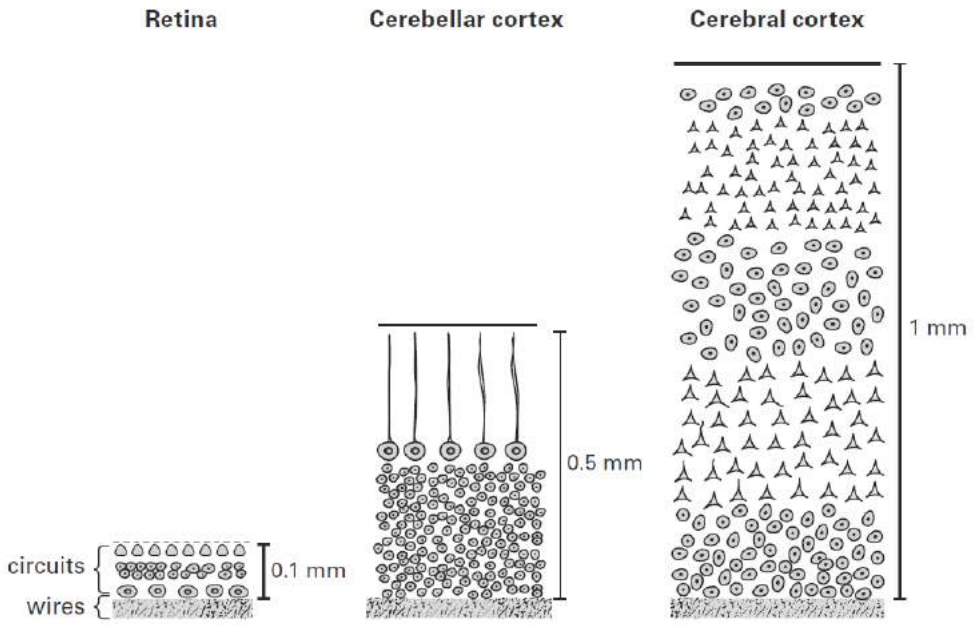
Patrones arquitectónicos
Capas

No sólo en software…

Córtex cerebral



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;}
};
```

Patrones arquitectónicos
Capas

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
```

Capas

Ventajas

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

Capas

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 puede que se realicen de forma innecesaria (p.ej. comprobación de errores).

Capas

Variante:
Sistema basado en capas "relajado"

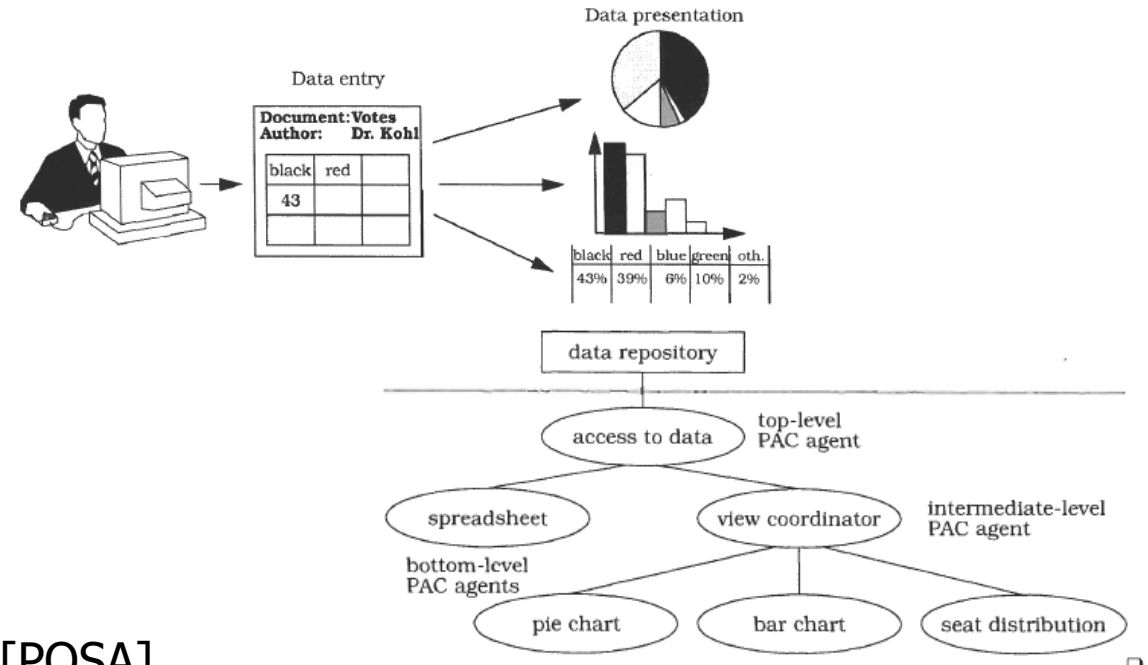
Cada capa puede utilizar los servicios
de todas las capas que quedan debajo de ella.

- Mayor flexibilidad y rendimiento.
- Menor mantenibilidad.

p.ej. Sistemas operativos UNIX

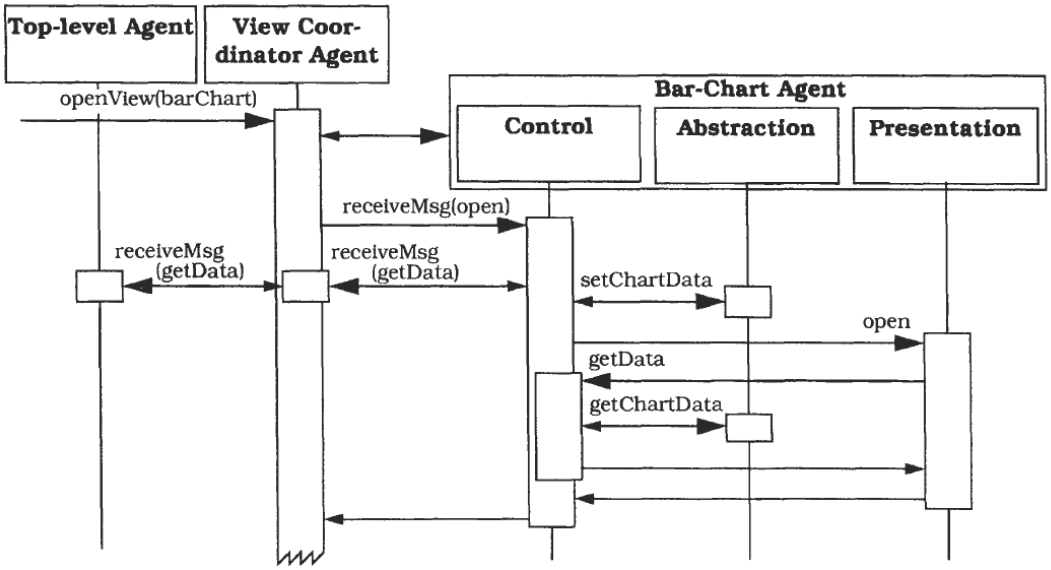
Capas

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 Microkernel	***Collaborators*** • Internal Server
Responsibility • Provides core mechanisms. • Offers communi-cation facilities. • Encapsulates sys-tem dependencies. • Manages and controls resources.	

[POSA]

Variante: Microkernel

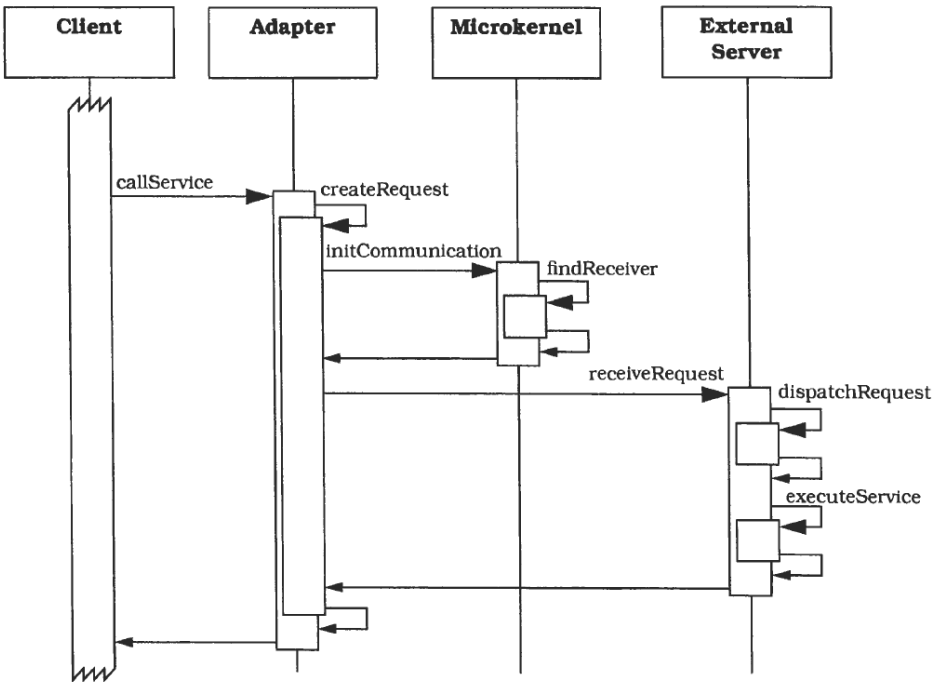
Class: Internal Server	Collaborators: Microkernel
Responsibility • Implements additional services. • Encapsulates some system specifics.	

Class: External Server	Collaborators: Microkernel
Responsibility • Provides programming interfaces for its clients.	

Class: Client	Collaborators: Adapter
Responsibility • Represents an application.	

Class: Adapter	Collaborators: External Server, Microkernel
Responsibility • Hides system dependencies such as communication facilities from the client. • Invokes methods of external servers on behalf of clients.	

Variante: Microkernel

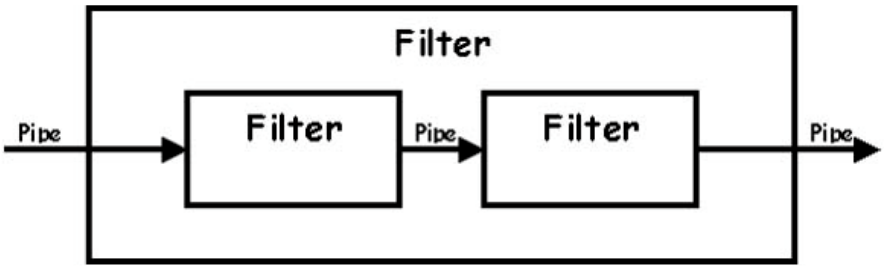


Patrones arquitectónicos
Flujos de datos [pipes & filters]

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]

Class Filter	Collaborators • Pipe
Responsibility • Gets input data. • Performs a function on its input data. • Supplies output data.	

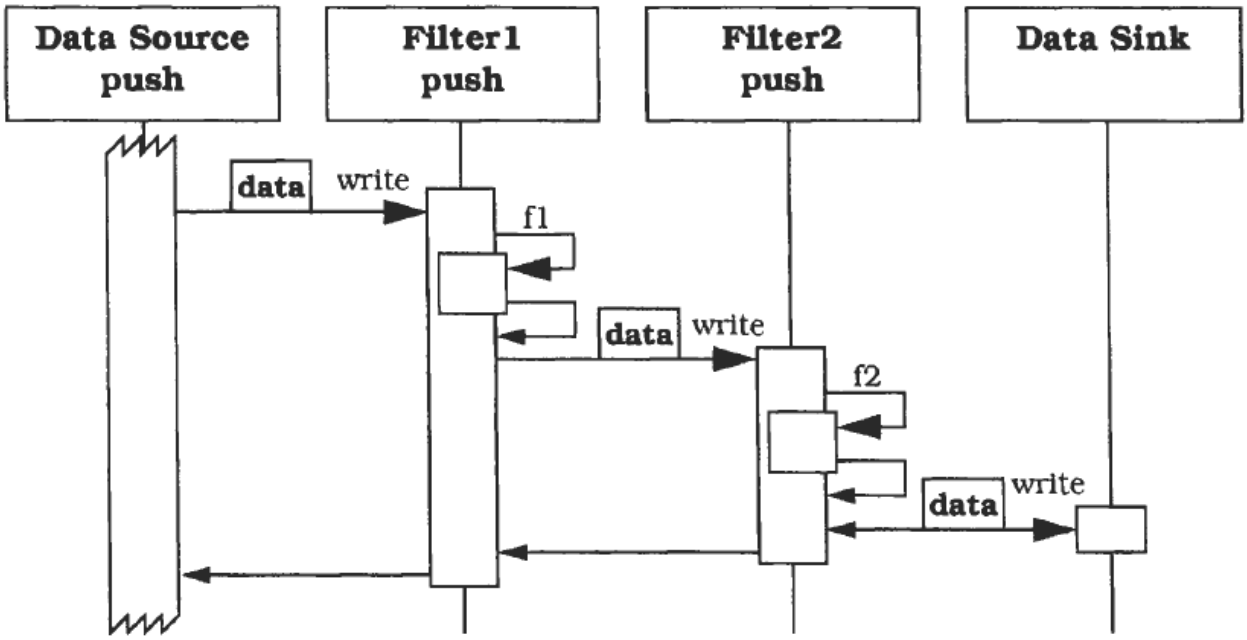
Class Pipe	Collaborators • Data Source • Data Sink • Filter
Responsibility • Transfers data. • Buffers data. • Synchronizes active neighbors.	

Class Data Source	Collaborators • Pipe
Responsibility • Delivers input to processing pipeline.	

Class Data Sink	Collaborators • Pipe
Responsibility • Consumes output.	

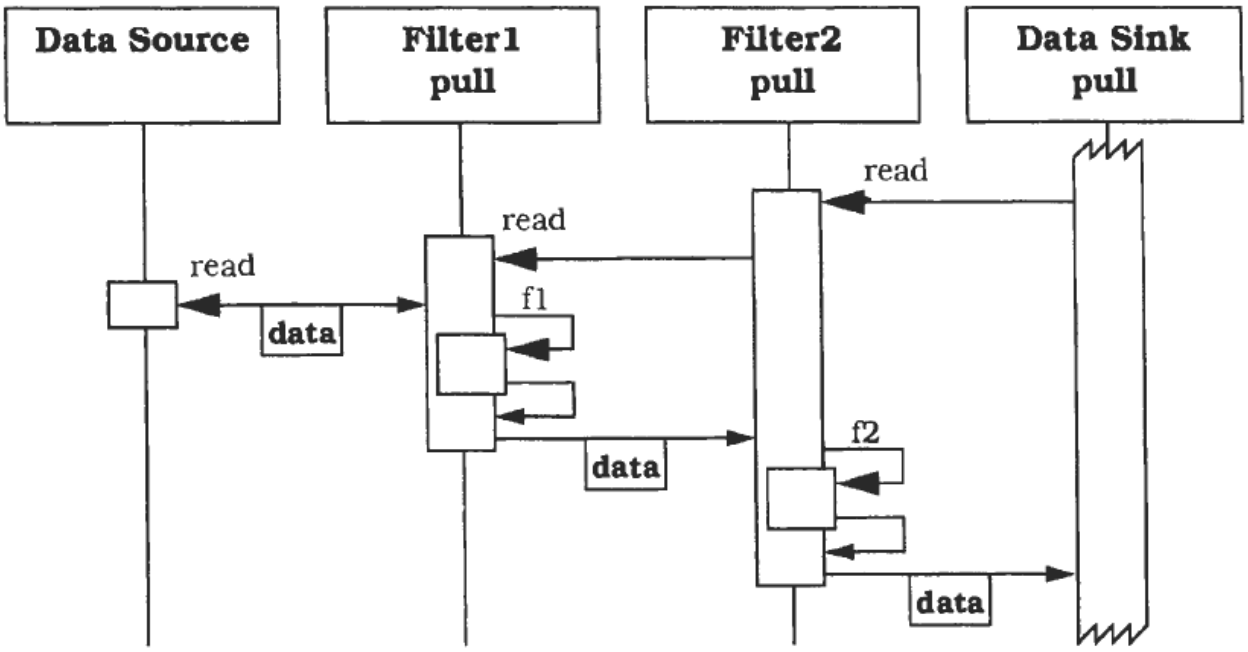
Flujos de datos [pipes & filters]

Push pipeline



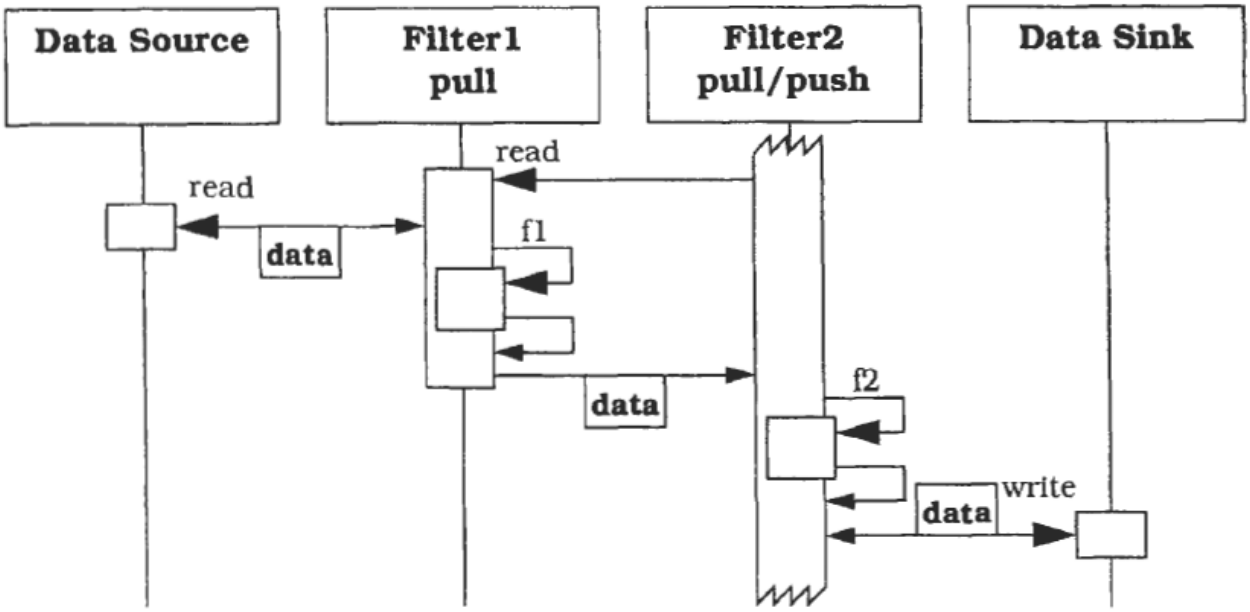
Flujos de datos [pipes & filters]

Pull pipeline



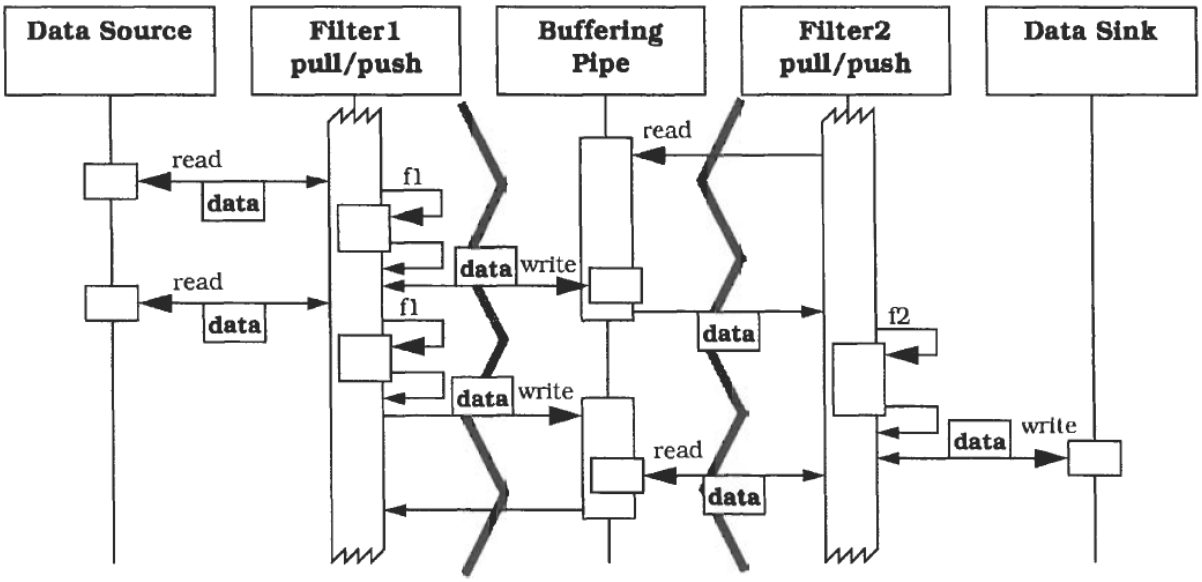
Flujos de datos [pipes & filters]

Mixed pipeline



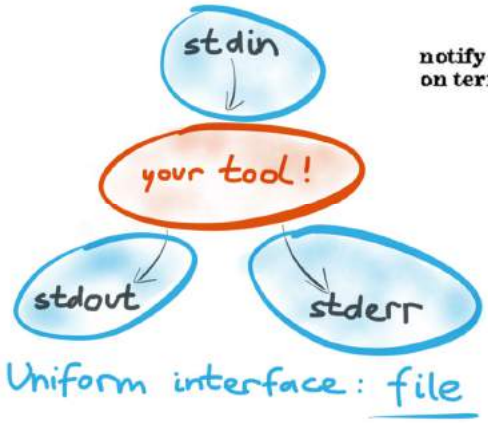
Flujos de datos [pipes & filters]

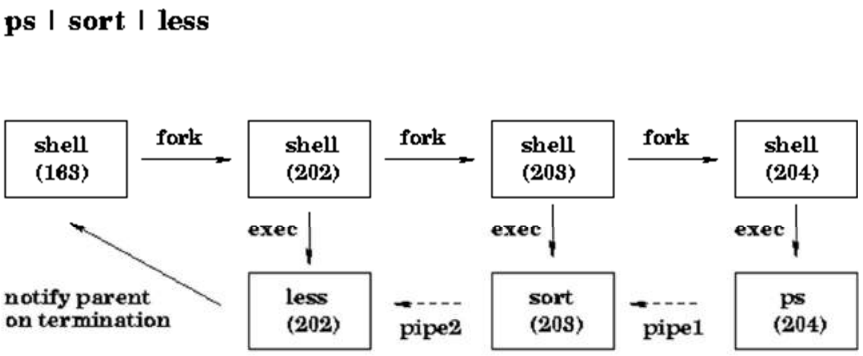
Pipes & filter system



Flujos de datos [pipes & filters]

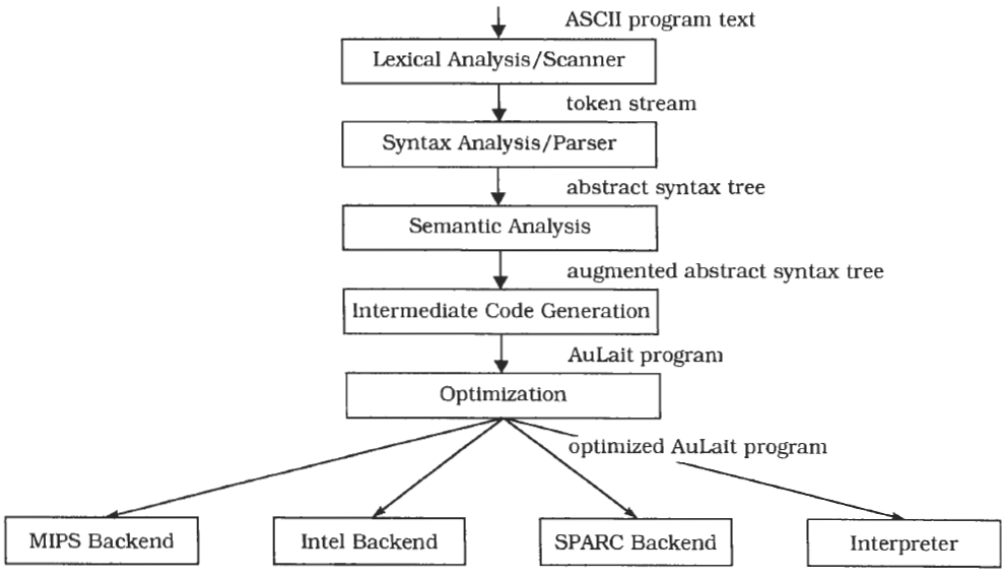
Herramientas UNIX





Flujos de datos [pipes & filters]

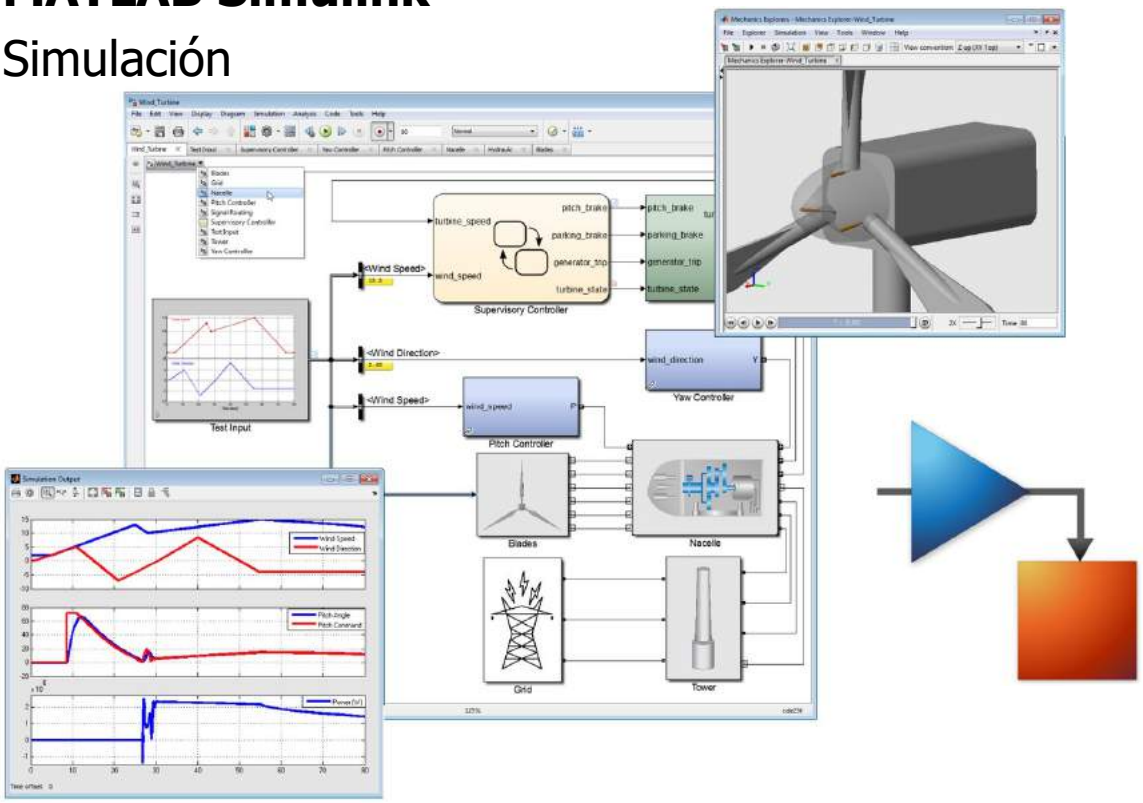
Intérpretes y compiladores



Flujos de datos [pipes & filters]

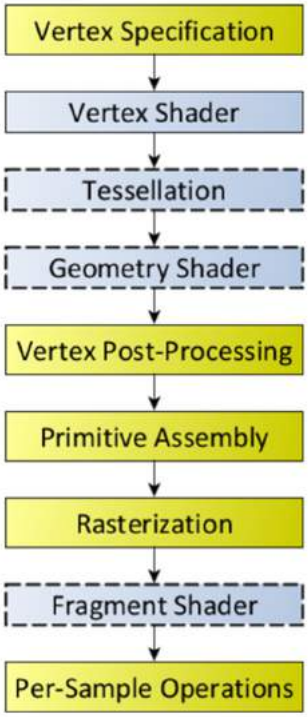
MATLAB Simulink

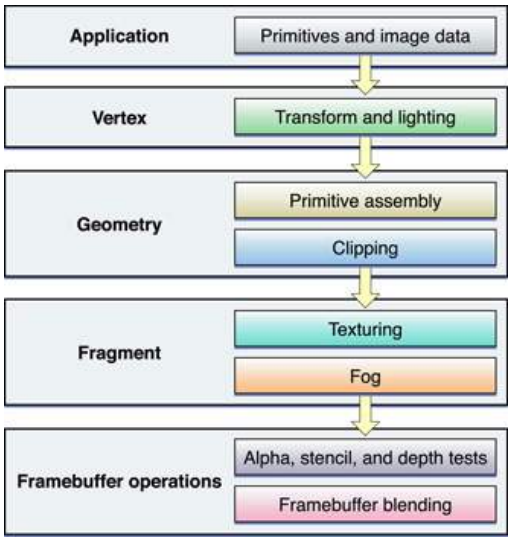
Simulación



Flujos de datos [pipes & filters]

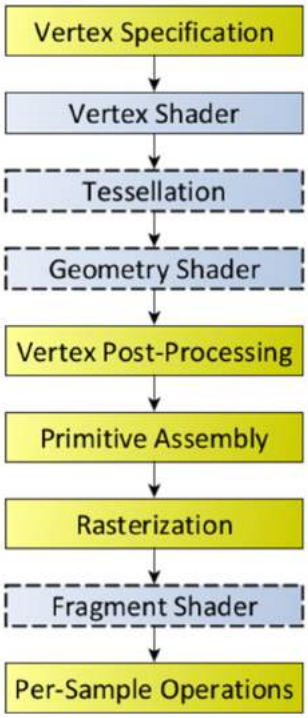
OpenGL rendering pipeline

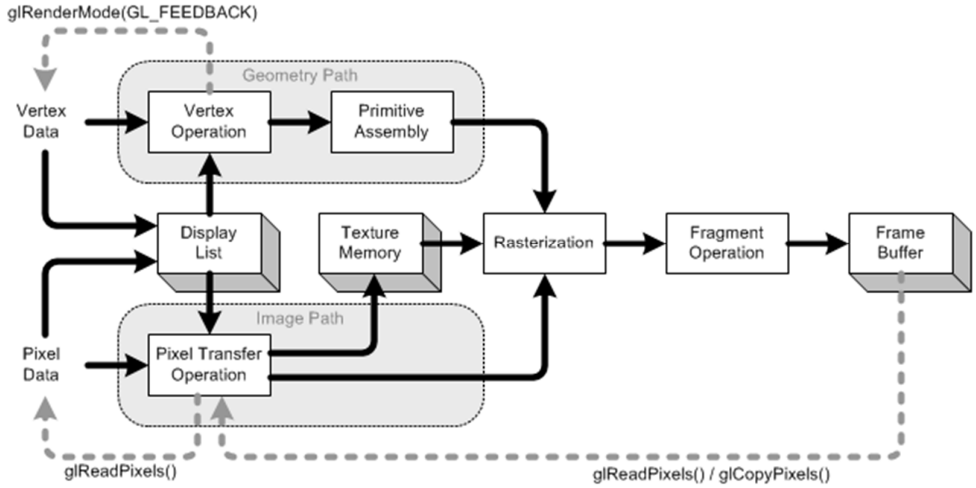




Flujos de datos [pipes & filters]

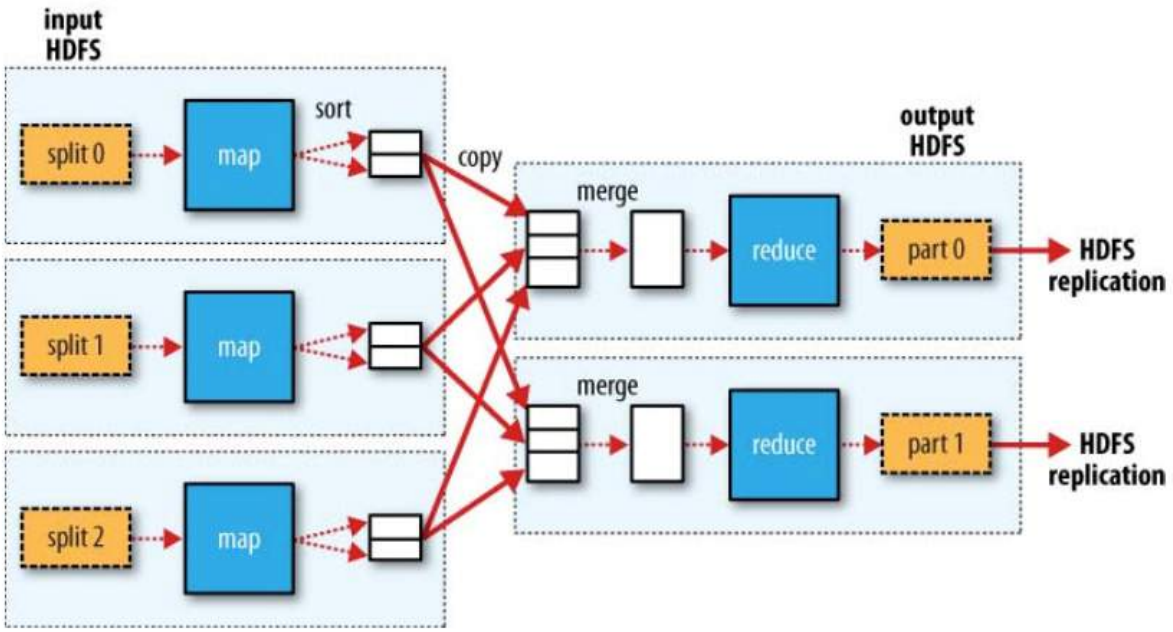
OpenGL rendering pipeline





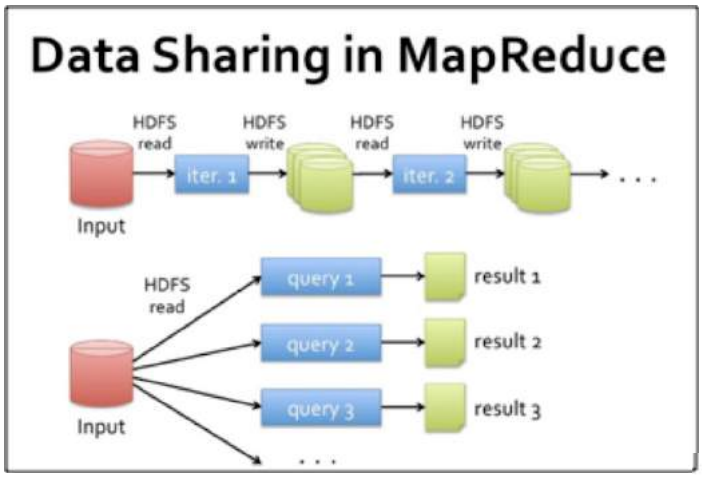
Flujos de datos [pipes & filters]

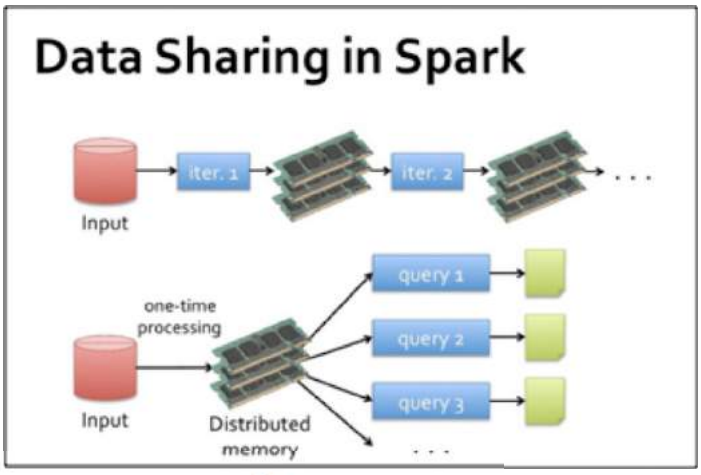
MapReduce



Flujos de datos [pipes & filters]

MapReduce

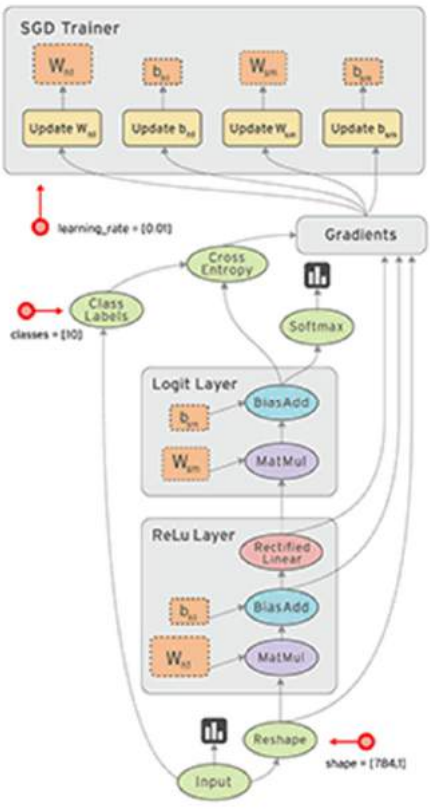




Flujos de datos [pipes & filters]

Google TensorFlow

https://www.tensorflow.org/

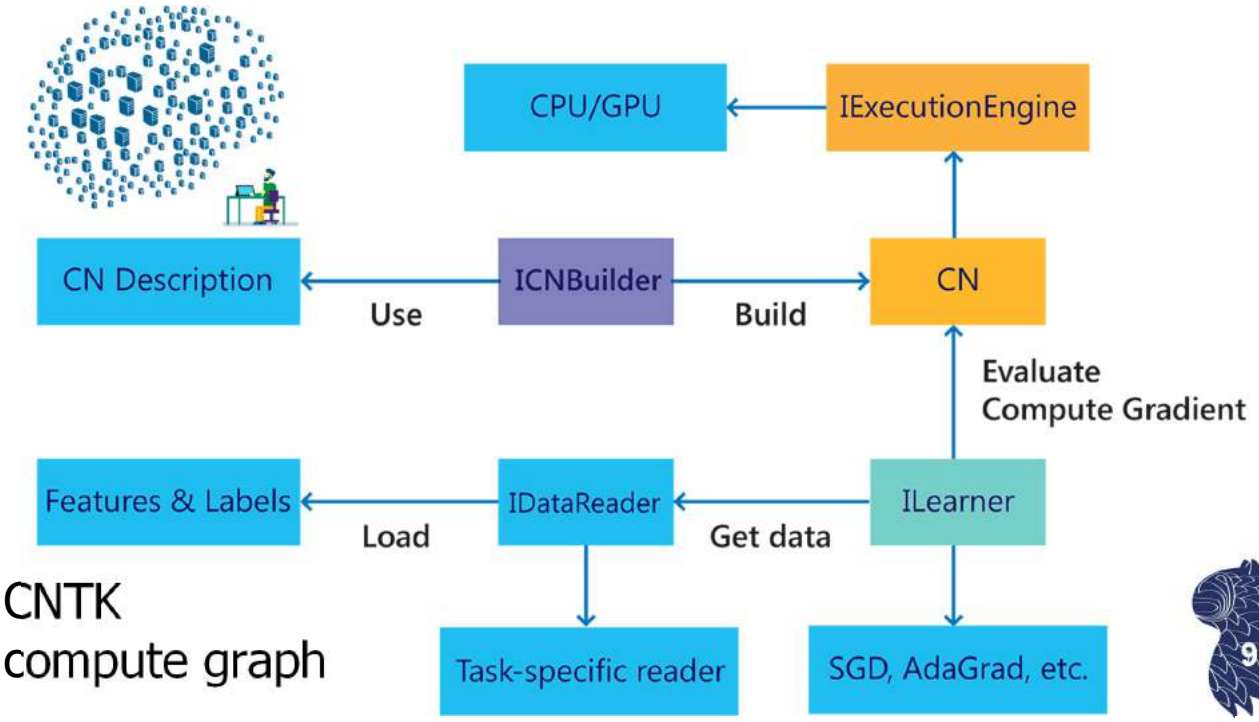


Data flow graph

Flujos de datos [pipes & filters]

Microsoft CNTK [Computational Network Toolkit]

http://www.cntk.ai/



CNTK
compute graph

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.

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)

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.

Flujos de datos [pipes & filters]

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.

Patrones arquitectónicos

Flujos de datos [pipes & filters]

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.

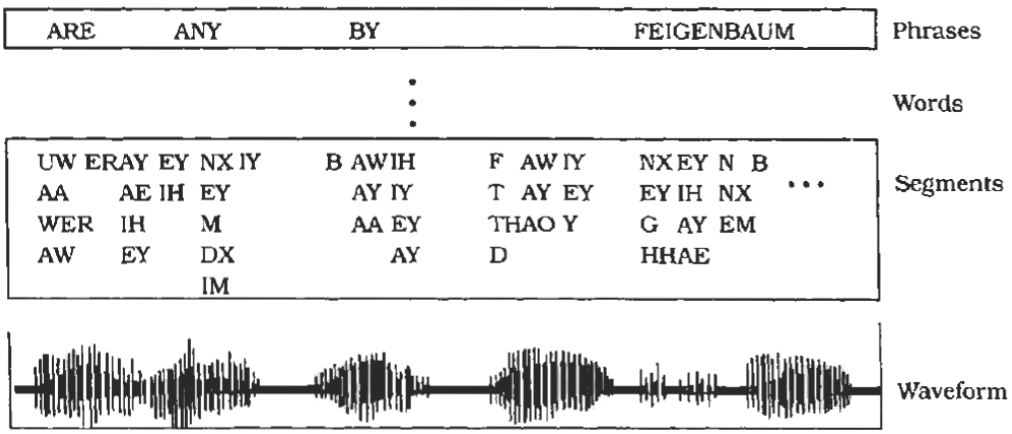
Patrones arquitectónicos
Pizarra

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

Patrones arquitectónicos
Pizarra

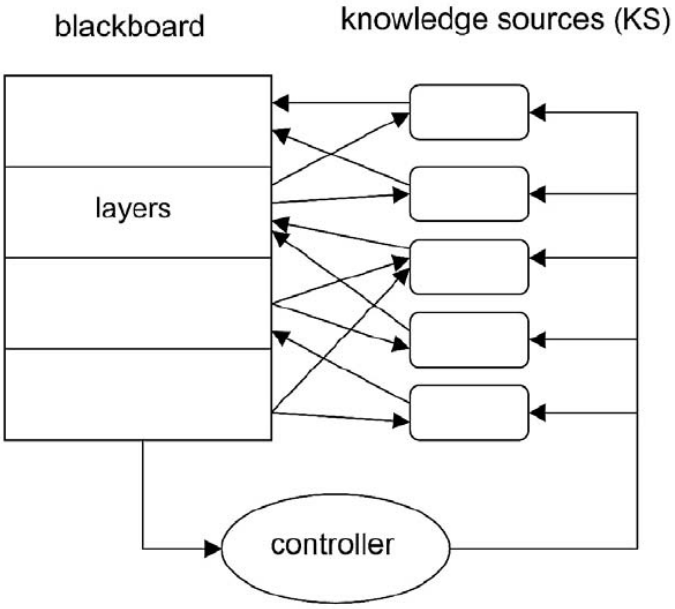
HEARSAY-II

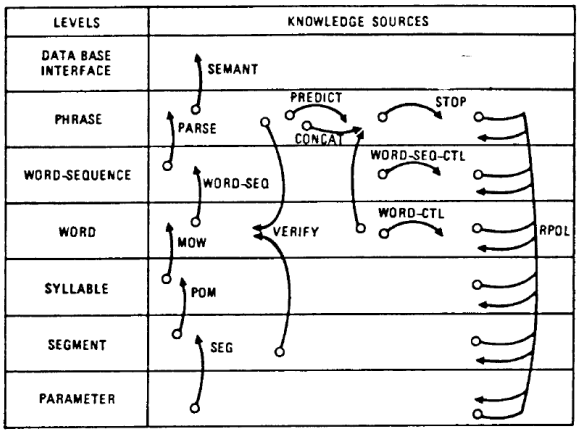
Reconocimiento de voz



HEARSAY-II

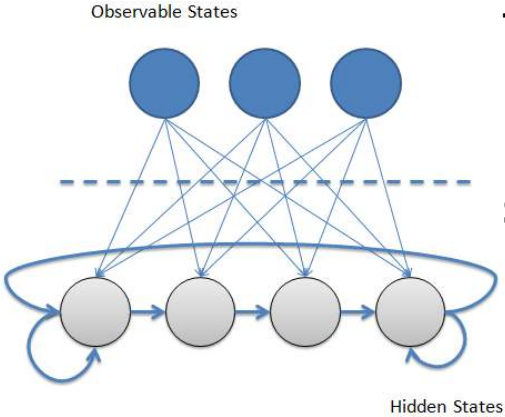
Reconocimiento de voz





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).

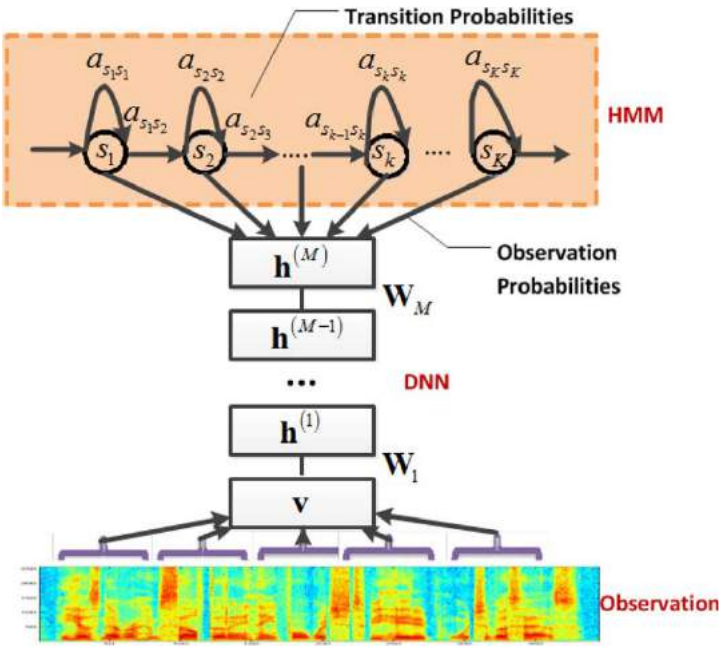


Técnicas estadísticas como los modelos ocultos de Markov [HMM] reemplazaron a los sistemas simbólicos como Hearsay-II.

Evolución de los sistemas de reconocimiento de voz

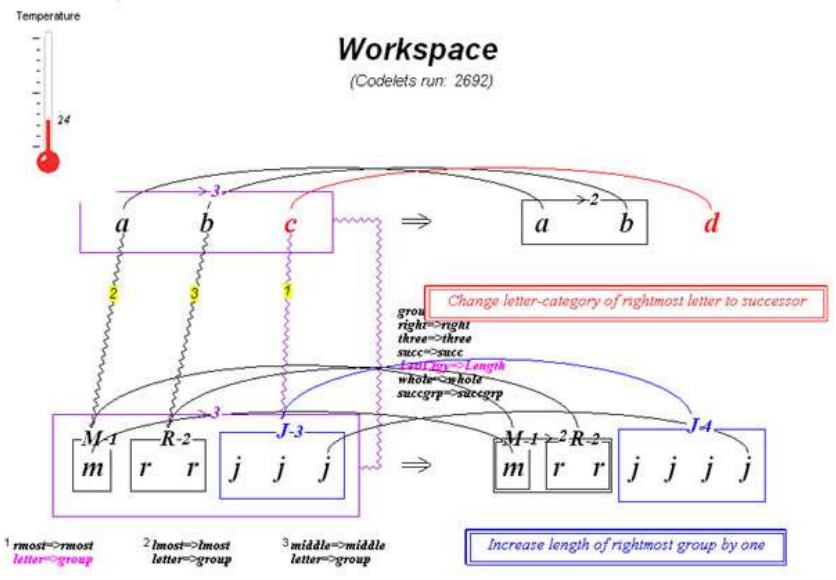
Actualmente, se siguen utilizando HMMs cuyos parámetros se ajustan con redes neuronales [deep learning].

Microsoft Research



Copycat

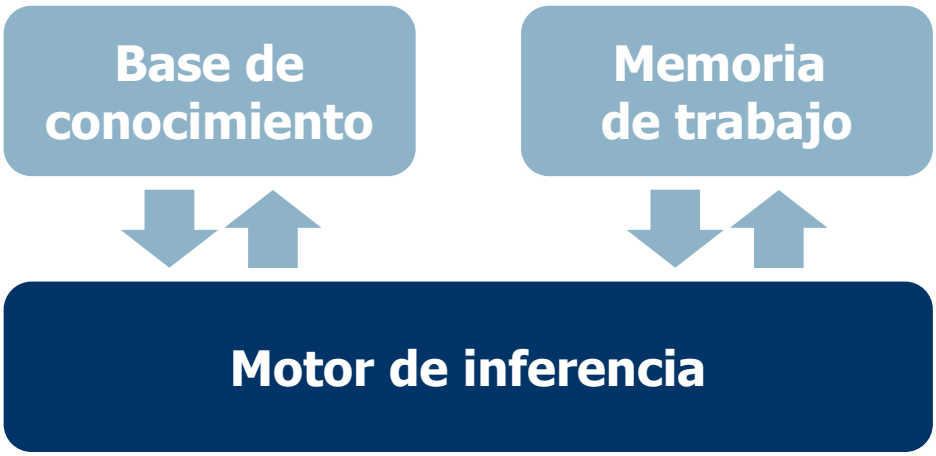
Razonamiento mediante analogías



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".

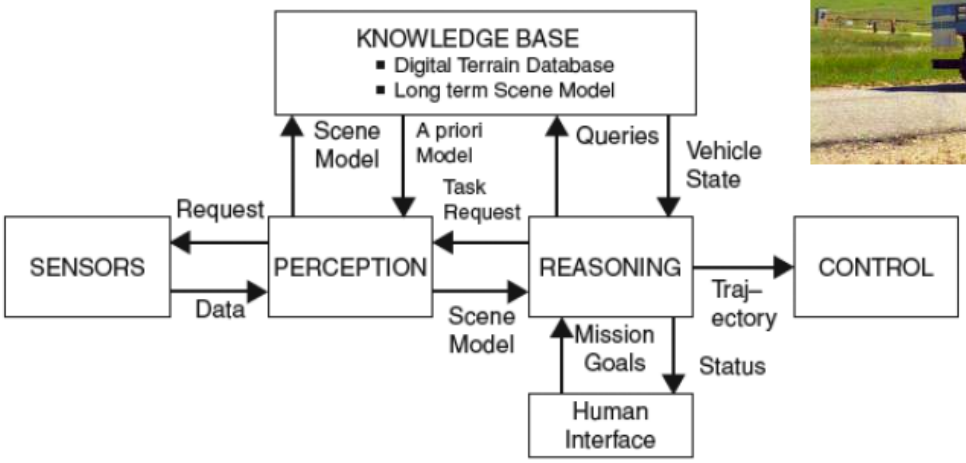
Sistemas expertos

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



ALV

Autonomous Land Vehicle



Pizarra

C4ISTAR

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

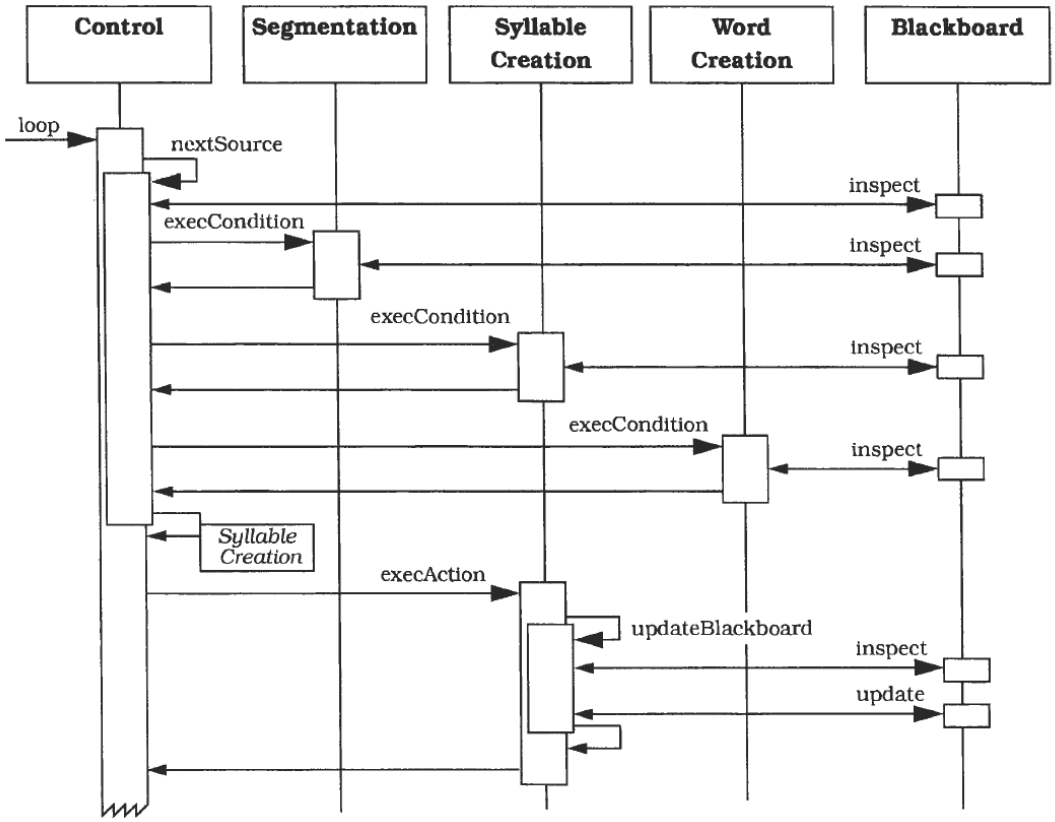
Sistemas militares de
detección y seguimiento de objetos

Pizarra

Tarjetas CRC

Class Blackboard	***Collaborators*** -
Responsibility • Manages central data	

Class Knowledge Source	***Collaborator*** • Blackboard
Responsibility • Evaluates its own applicability • Computes a result • Updates Black-board	



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.

Pizarra

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.

Pizarra

Variantes

- **Sistemas de producción**

  p.ej. Business rules engines
  https://en.wikipedia.org/wiki/Business_rules_engine

- **Repositorios**
  (sin mecanismo de control centralizado)

  p.ej. Bases de datos
  IDEs

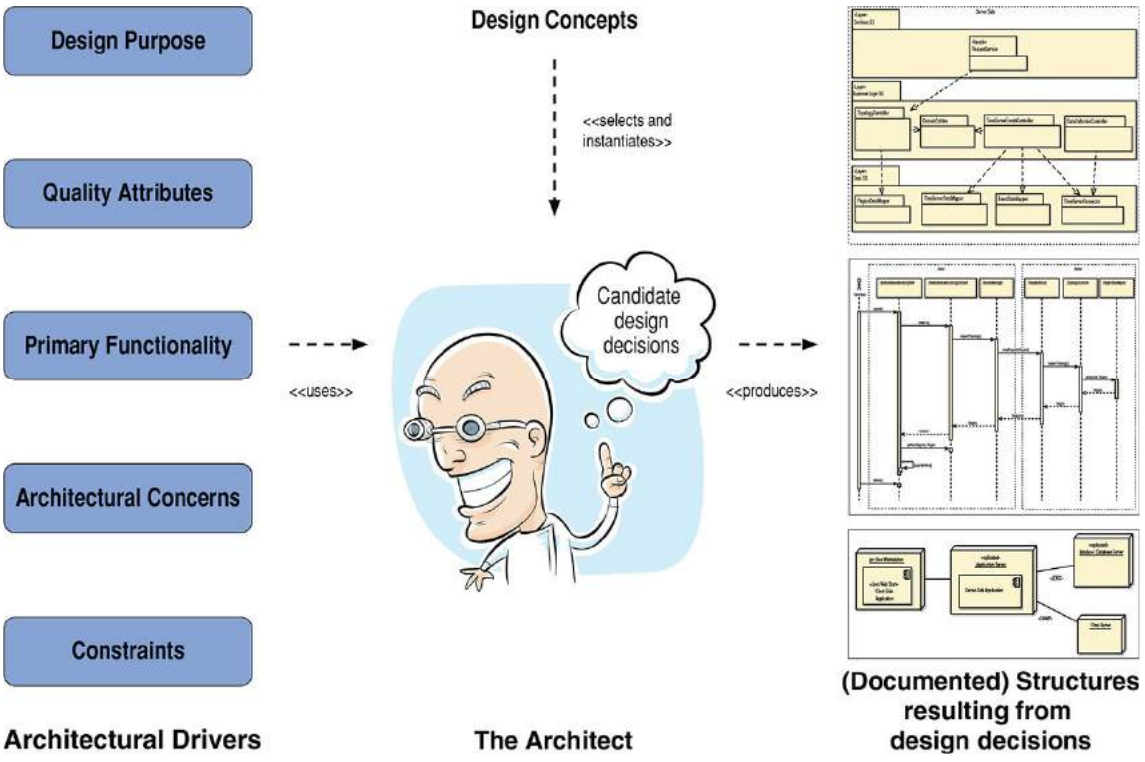
Ventajas

- 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)

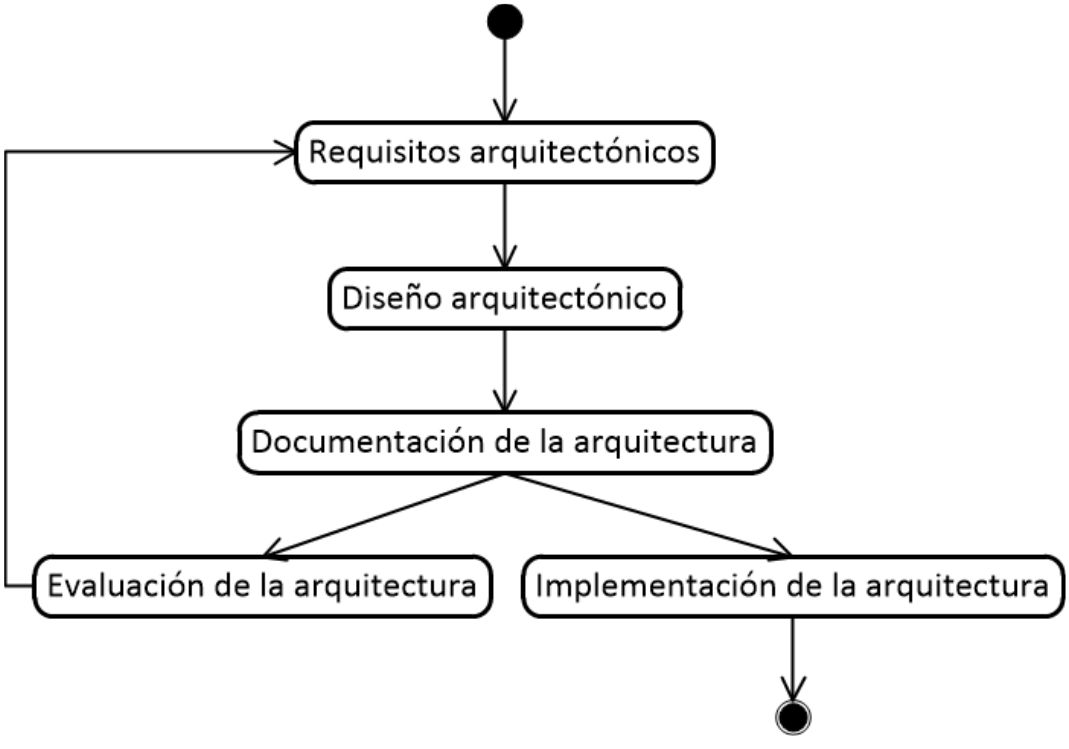
Consecuencias

- 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

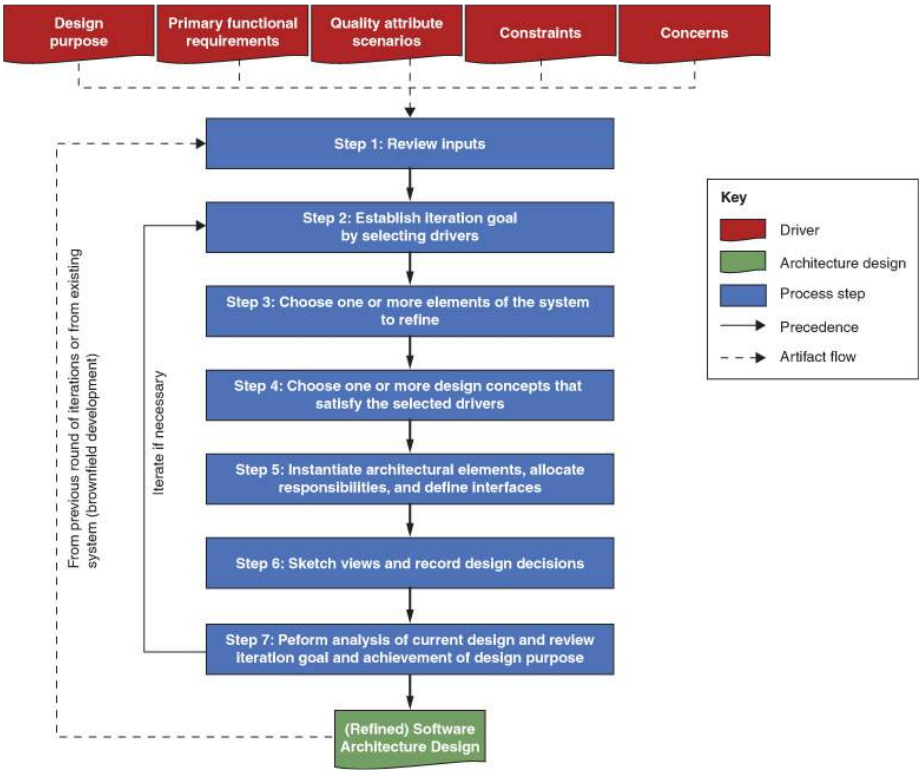


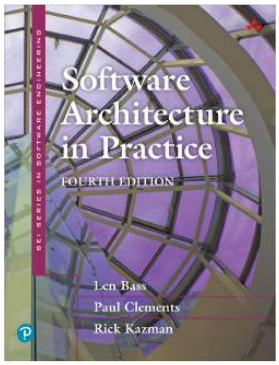
Diseño arquitectónico
El proceso de diseño



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?

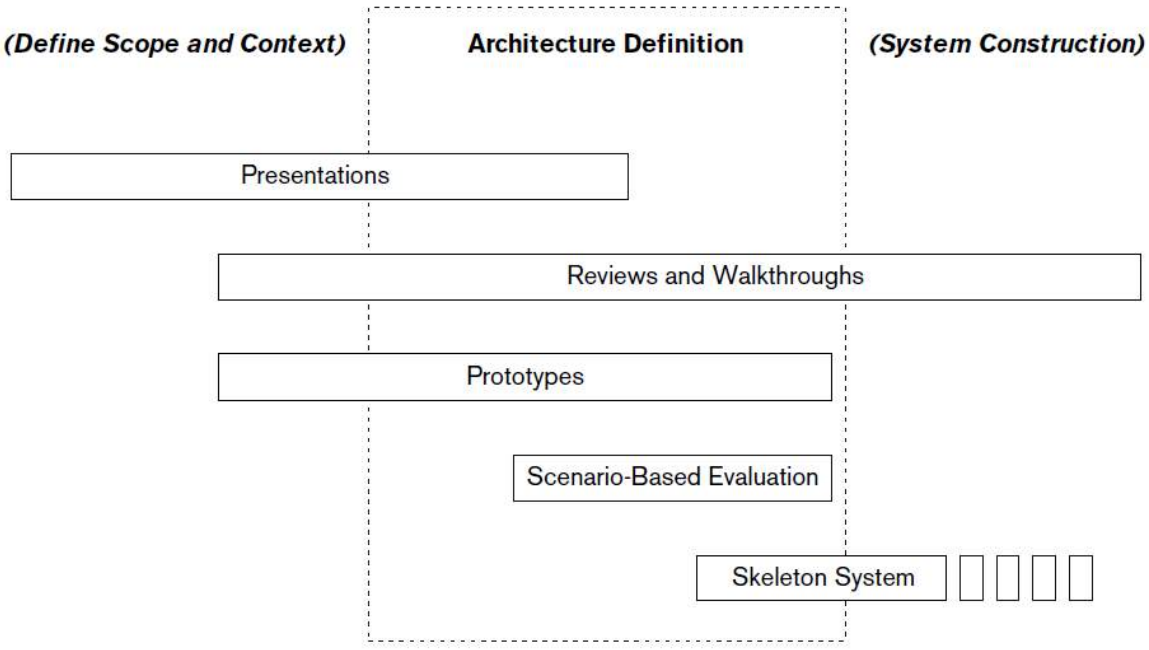
Validación de la arquitectura

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



Validación de la arquitectura

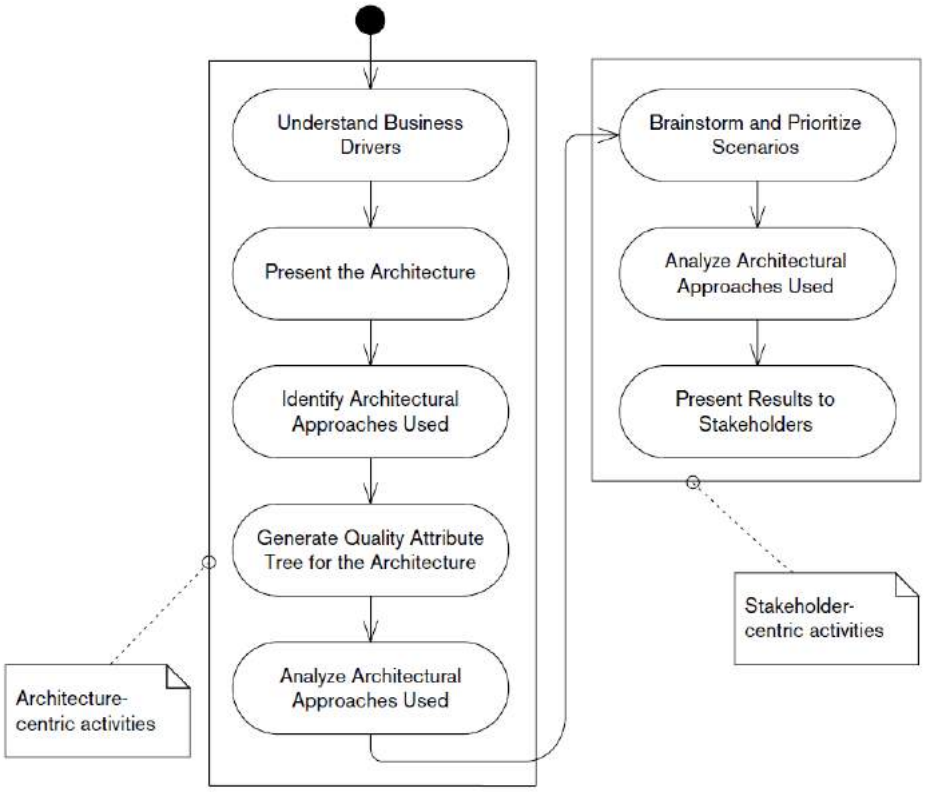
Evaluación mediante escenarios

- **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.

Validación de la arquitectura

Evaluación mediante escenarios

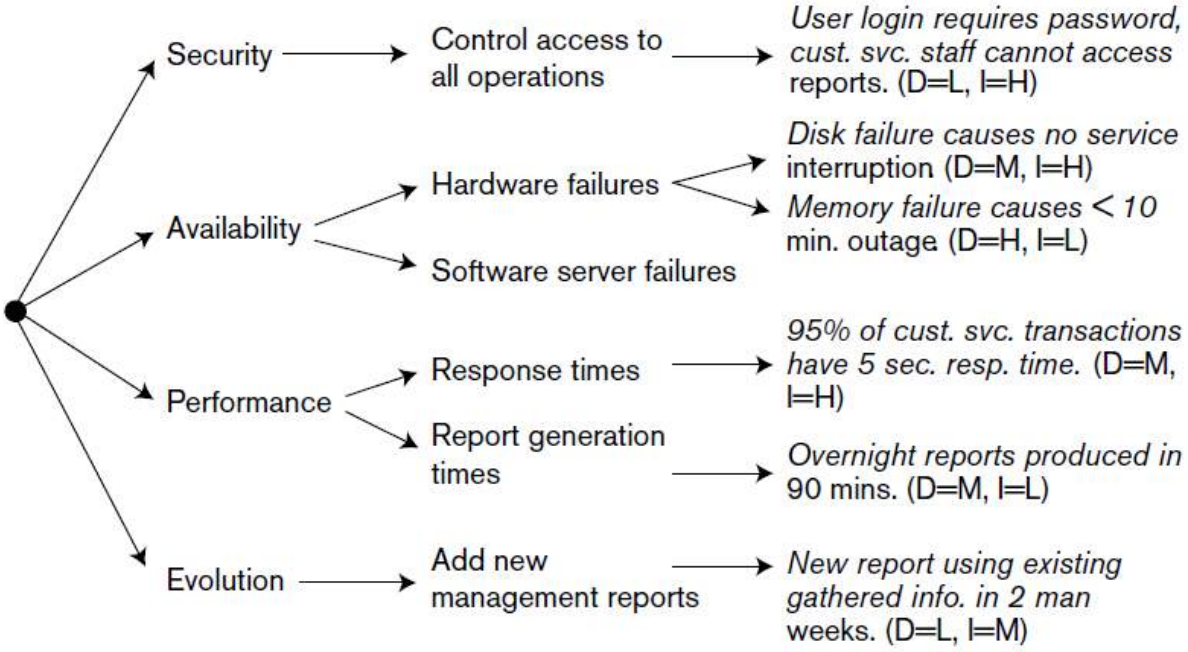
ATAM



Validación de la arquitectura

Evaluación mediante escenarios

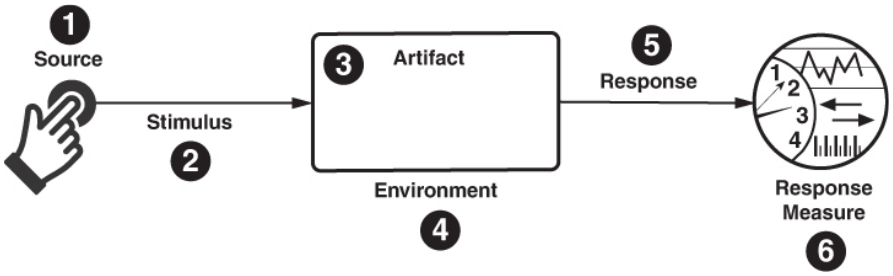
ATAM Quality Attribute Tree, a.k.a. utility tree

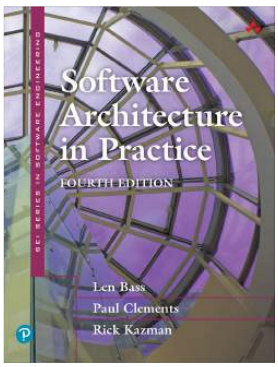


Validación de la arquitectura

Evaluación mediante escenarios

Atributos de calidad [Quality Attributes]



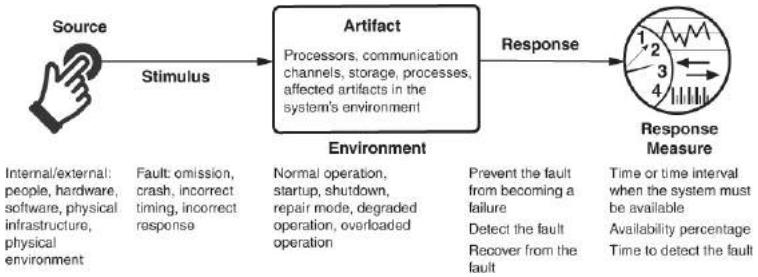


IDEA CLAVE:
Especificación testable, libre de ambigüedades.

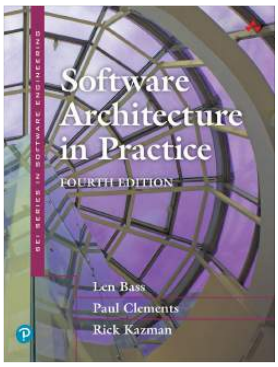
Validación de la arquitectura

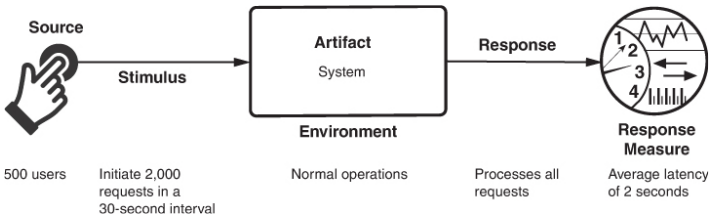
Evaluación mediante escenarios

Ejemplos



DISPONIBILIDAD [AVAILABILITY]



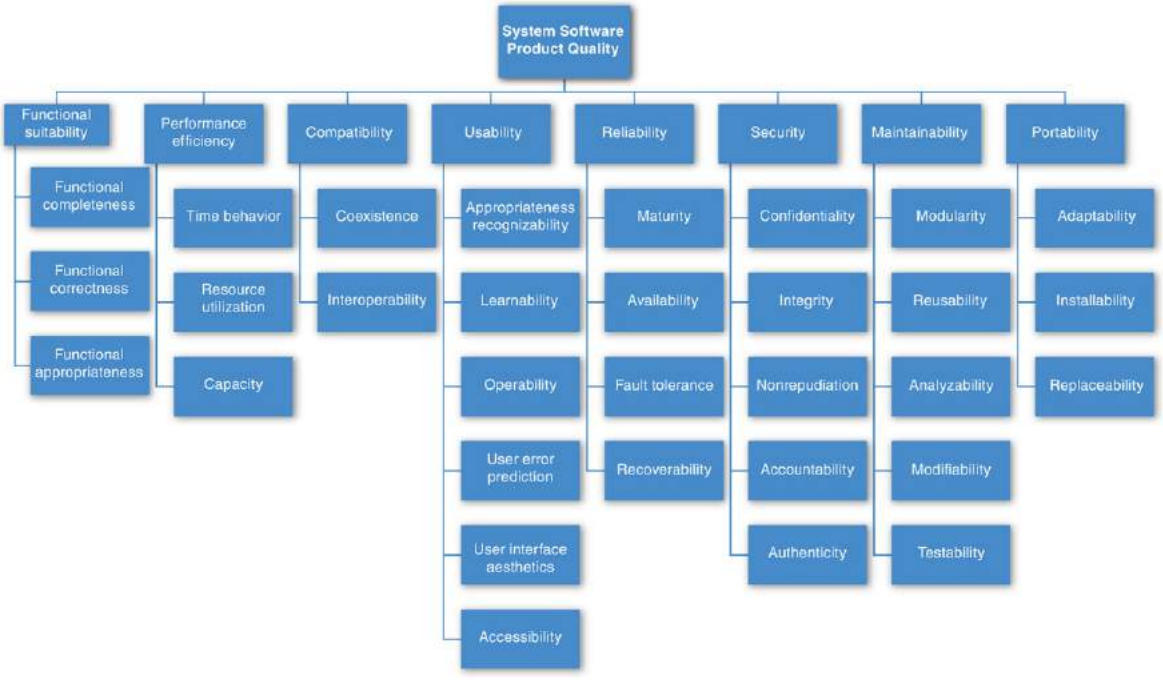


RENDIMIENTO [PERFORMANCE]

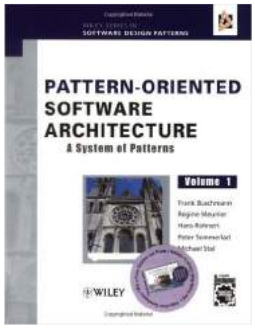
Validación de la arquitectura

Evaluación mediante escenarios

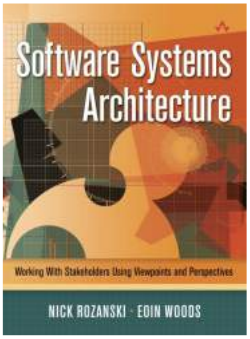
Atributos de calidad, estándar ISO/IEC FCD 25010



Bibliografía recomendada

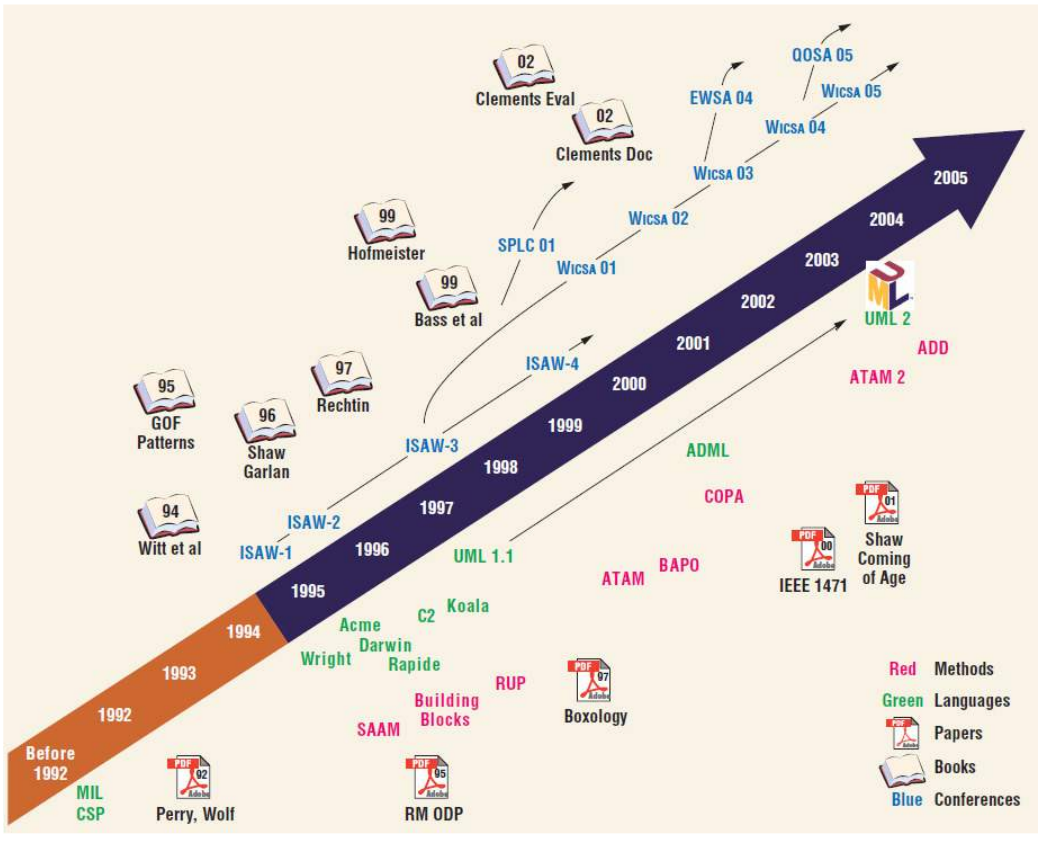


- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad & Michael Stal: **Pattern-Oriented Software Architecture. Volume 1: A System of Patterns** Wiley, 1996. ISBN 0471958697



- Nick Rozanski & Eóin Woods: **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives** Addison-Wesley Professional, 2005. ISBN 0321112296

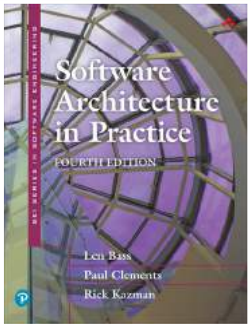
Bibliografía



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)

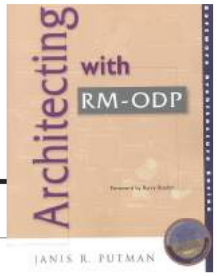
COMPORTAMIENTO

- 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

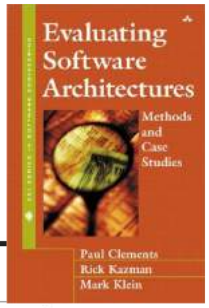
Christine Hofmeister, Robert Nord & Dilip Soni:
Applied Software Architecture
Addison-Wesley, 2000

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.

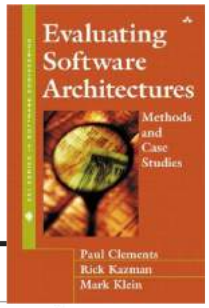


Viewtype	Definition
Component and Connector	The Component and Connector viewtype is concerned with the system's runtime functional elements, their behaviors, and their interactions. The following styles defined for this viewtype all relate to commonly occurring runtime system organizations: • *Pipe-and-Filter* • *Shared-Data* • *Publish-Subscribe* • *Client-Server* • *Peer-to-Peer* • *Communicating-Processes*

Apéndice: Conjuntos de vistas

SEI Viewtypes

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

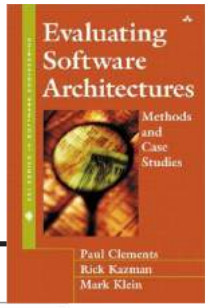


Viewtype	Definition
Module	The Module viewtype is concerned with how the software comprising the system is structured as a set of implementation (code) units. The following styles are defined for the Module viewtype: • *Uses*: for capturing intermodule usage dependencies • *Generalization*: for capturing commonality and variation (inheritance) relationships between modules • *Decomposition*: for specifying how modules are composed from simpler elements • *Layered*: for specifying how modules are arranged in layers according to their level of abstraction

Apéndice: Conjuntos de vistas

SEI Viewtypes

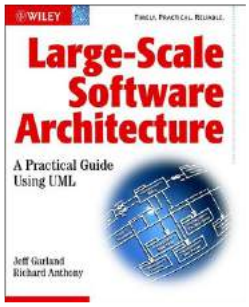
Paul Clements, Rick Kazman & Mark Klein:
Evaluating Software Architectures.
Addison-Wesley, 2002.



Viewtype	Definition
Allocation	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: • *Deployment*: for specifying how software elements are mapped to elements of the deployment environment • *Implementation*: for specifying how software modules are mapped to the development environment (such as their location in a codeline) • *Work Assignment*: for mapping software modules to those responsible for creating, testing, and deploying them

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