Introducción a la Arquitectura de Software

Gustavo Rossi

Contexto: Que software desarrollamos hoy

- Gran cantidad de empresas cuyo mayor (único?) activo es el software: Glovo, Uber, Booking, Facebook, Google
- Aplicaciones Distribuidas que combinan Software, Comunicaciones, Sensores, Hardware, Personas, todas ellas en dominios novedosos
- Cantidad creciente de funcionalidad "importada" de terceros, desde servicios específicos de bajo nivel (Cloud), o alto nivel (funcionalidad multimedia, redes sociales, etc)
- "Destilación" permanente de datos mediante análisis de grandes volúmenes de información para cambiar el comportamiento general de las aplicaciones

Que tipo de aplicaciones

- Aplicaciones que combinan “partes” de otras aplicaciones. El reuso pasó a ser imprescindible. Se diseña para reusar.
- Que pueden crecer en forma completamente ininmaginable..
- Que se componen y descomponen y cuyas partes son usadas por otros….
- **Con Requerimientos no-funcionales “nuevos”**
- “Nuevos” dominios y problemas: Aplicaciones inter-vehiculares, Integración de Big Data e IA en software “convencional”, etc

Que nuevos problemas nos generan?

- "Divide and Conquer" complejizado a niveles nunca inmaginados
- Los módulos (componentes, clases, servicios, etc) de nuestra aplicación ya no co-existen (incluso ya no lo hacen "amigablemente") en la misma plataforma
- Los módulos pueden estar desarrollados en diferentes lenguajes, comunicarse via diferentes mecanismos, obviamente estar desarrollados por equipos diferentes, quizás sin relación entre ellos.
- El problema de búsqueda de modularidad para asegurar mejor mantenimiento obviamente se complica
- **Debemos preocuparnos por "nuevos" requerimientos no funcionales: escalabilidad, disponibilidad, usabilidad, seguridad, privacidad, performance, aspectos legales....**

Como los resolvemos?

- Teniendo buenas estrategias de separación de concerns para atacar problemas diferentes en módulos diferentes
- Aprendiendo los conceptos que nos permiten hacer "divide and conquer" en diferentes niveles
- Aprendiendo mecanismos de interoperabilidad y "sharing" de información entre módulos y aplicaciones
- **Conociendo los requerimientos no-funcionales mas usuales (escalabilidad, seguridad, usabilidad....)**
- Conociendo estilos y ejemplos de arquitecturas exitosas

Niveles de Diseño

- Igual que en el Diseño arquitectural, en software también tenemos diferentes niveles y problemas de diseño, de diferente granularidad
- Diseño de la colaboración entre objetos, diseño de estructuras de información, diseño de la relación y comunicación entre componentes, etc.

La arquitectura del software

- Depende fuertemente de los requisitos no-funcionales
- Decidimos usar una u otra arquitectura en base a cuales requisitos queremos satisfacer

- La arquitectura de software define y es previa, en general, a la de hardware

Breve historia de la evolucion

- Previo a 1985: Diseño estructurado, Diseño OO "intuitivo", Redes "ad-hoc"
- 1991: CORBA
- Mid-90: Técnicas de Diseño OO, Patrones e Introducción al concepto de Arquitectura y Estilos arquitecturales
- Fin de los ´90: SOA (Service Oriented Architecture)
- 2000-2005: Web Services/REST
- 2005: Microservices, Cloud
- 2010: Serverless Computing

Muchas de las definiciones están "influidas" por el contexto tecnológico

Arquitectura de Software, que es?

- **Shaw and Garlan (‘96)**: Software architecture describes the structural issues of a software system, including its organization as a composition of components, communication protocols, synchronization, data access, assigning of functionality to design elements, physical distribution

Que es?

- **IEEE Definition**: Architecture is the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces

Que es?

- **Keeling(2017):** system's software architecture is the set of significant design decisions about how the software is organized to promote desired quality attributes and other properties. A design decision might be significant for any number of reasons. It might represent a point of no return or influence quality attributes, schedule, or costs. A significant decision might be one that affects many people or forces other software systems to change. In any case, significant design decisions are costly to change later if you get them wrong.

Que es?

- **Ford (2020):** Software architecture consists of the structure of the system, combined with architecture characteristics (“-ilities”) the system must support, architecture decisions, and finally design principles. The structure of the system refers to the type of architecture style (or styles) the system is implemented in (such as microservices, layered, or microkernel). The architecture characteristics define the success criteria of a system, which is generally orthogonal to the functionality of the system (availability, security, scalability.....). Architecture decisions define the rules for how a system should be constructed. For example, an architect might make an architecture decision that only the business and services layers within a layered architecture can access the database restricting the presentation layer from making direct database calls. Architecture decisions form the constraints of the system and direct the development teams on what is and what isn’t allowed.

Diseño de Arquitecturas de Software

- Muy influenciado por la selección de alguna arquitectura o estilo arquitectural "de referencia" (e.g. Cliente/Servidor, Servicios, etc)
- Tres tipos de problemas:
  - Diseño Arquitectural (definición de los elementos "fundamentales" de nuestro sistema)
  - Diseño de la interacción entre elementos (componentes, partes)
  - Diseño "interno" de los elementos
- Obsérvese que la complejidad de cada una de estas actividades tiene una relación estrecha con el estilo de arquitectura elegida

Estilos (o patrones) arquitecturales

- Representan soluciones genéricas y reusables en muy alto nivel a problemas recurrentes en arquitectura de software
- Parecidas conceptualmente a los estilos en arquitectura
- Todos entendemos a que se refiere el concepto edificio en torre, o barrio privado, o chalet, más allá de los detalles
- En software tenemos algo semejante: cliente/servidor o arquitectura en capas, o arquitectura basada en servicios, etc.

Estilos Arquitecturales

- Evolucionaron en popularidad desde los 90 hasta hoy
- En la década del 90 los estilos preferidos eran: Client-Server, Layered y Blackboard
- Hacia fines de los 90 aparecen Publish/Subscribe (en el contexto de Componentes), MVC, ORB con RPC
- Las arquitecturas basadas en servicios se vuelven populares en 2006 y las de microservicios en 2015
- Cada uno de los patrones y estilos de arquitectura tienen un impacto positivo en algún atributo de calidad y eventualmente uno negativo en otro

por

Year	CLIENT-SERVER	LAYERS	PIPES AND FILTERS	SOA	MVC	COMPONENT-BASED	BLACKBOARD	PUBLISH-SUBSCRIBE	C2	IMPLICIT INVOCATION	BROKER	SHARED REPOSITORY	SPACE-BASED	PEER-TO-PEER	MICROSERVICE	PAC	MICROKERNEL	RPC	VIRTUAL MACHINE	REFLECTION	INTERPRETER	RULE-BASED SYSTEM	EXPLICIT INVOCATION	MASTER-SLAVE	BATCH SEQUENTIAL	INDIRECTION LAYER	INTERCEPTOR	MESSAGE QUEUING	CQRS	# studies per year
2019				1											1															2
2018	4	2	1	3		1		2					3	1	5											1		1	1	25
2017	1			7	2	1				1			2		9				1										1	25
2016	5	6	4	3	6	1	1	2	1	1	3	2	3	1	2	1	1			2		1							1	47
2015	7	6	3	3	3	4	2	1		1	3		5		1		1				1			1		1	1			44
2014	4	3	1	8	2	2		2		1	1	1	3			1	1	2										1		33
2013	2	1	2	4		1	1			1		2	1	1				1	1			1	1					1		21
2012	2	4	1	6	4		1						2			1					1		1							23
2011	5	6	5	6	3	3	3	2	1	1	1	3	1	3		1	1	1	1	1			2		1					51
2010	11	15	9	3	6		6	5	5	3	4	5		1		4	3	1	2	2	1			1						87
2009	1	1	4	5	1	1		2	3	1				2										1						22
2008	5	6	6	1	6		5	3	3	2	4	2		2		3	1		2		2		1			1	1			56
2007	5	4	2	4	3	3	1	3	1		1			2		2	2	1		1										35
2006	7	4	3	2	1	2		1	4		1			3				1		1					1		1			32
2005	5	3	1		2	2	1	2	2	1	1	2		3		1	1	1	1	1	1	1	1		1	1	1	1		37
2004	4	2			1	4		1	2		2	1																		17
2003	4		1		1	1		1								1								1						10
2002	3		1	1		1			1																					7
2001	1	1	2		1		1	1			1					1	2	1		1				1						14
2000	2	2	1		1		1			1	1					1	1	1		1										13
1999	3	3	4		1	4	1		3	2	1	1				1		1				1			1					27
1998	1		1							1																				3
1997	1		1						1																					3
1996		2	2		1	1	1		1	2								1	1		1									13
1995	6	3	5			1	3			3		1						2	1		1									26
1994			1							1																				2
1993		1	1				1			1		1							1		1				1					8
1992																						1								1
1991		1					1												1		1	1								5
1990	1																													1
# studies (patterns)	90	76	62	57	45	33	30	28	28	24	24	21	20	19	18	18	14	14	12	10	10	6	6	5	5	4	4	4	3	

Fuente:

Capturing Software Architecture Knowledge for Pattern-Driven Design
Siamak Farshidi, Slinger Jansen, Jan Martijn van der Werf

Journal of Systems and Software 2020

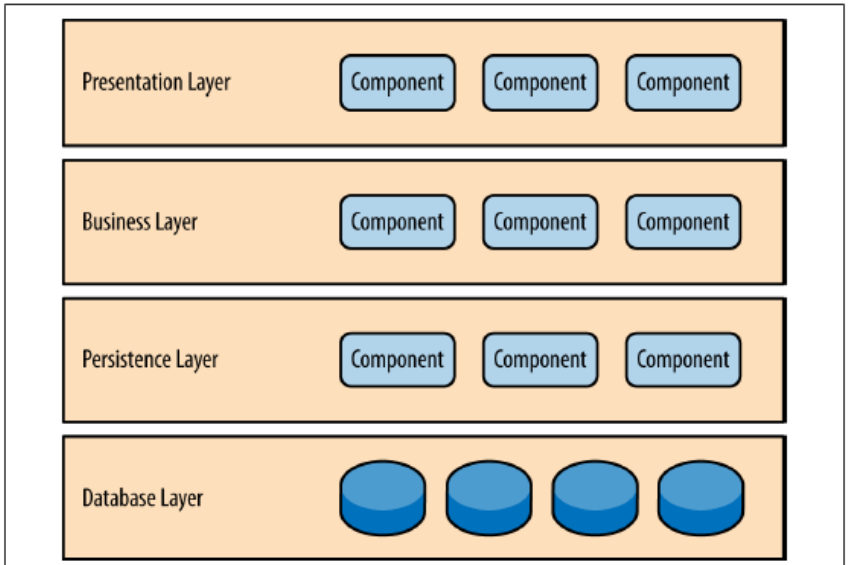
Quality Attributes	COMPONENT-BASED	SOA	PIPES AND FILTERS	LAYERS	IMPLICIT INVOCATION	CLIENT-SERVER	BROKER	BLACKBOARD	MVC	MICROKERNEL	MICROSERVICE	SPACE-BASED
Scalability	1|0|0 0,79	5|0|0 0,79	3|1|4 0,45	0|1|5 0,25	2|0|4 0,36	5|4|0 0,66	1|0|0 0,79	4|1|0 0,74	0|0|1 0,21	3|0|1 0,7	9|0|0 0,79	8|0|0 0,79
Reusability	10|0|0 0,79	19|1|0 0,77	11|2|1 0,73	16|2|0 0,76	7|1|0 0,76	3|2|1 0,61	4|0|0 0,79	6|2|1 0,69	2|1|1 0,6	3|0|0 0,79	3|0|0 0,79	
Performance efficiency	2|0|1 0,65	2|0|1 0,65	10|2|7 0,56	3|2|15 0,29	0|1|4 0,26	4|2|4 0,5	1|1|5 0,3	3|2|8 0,36	0|1|5 0,25	1|0|5 0,26	3|0|1 0,7	2|1|0 0,7
Portability	1|0|0 0,79	1|0|0 0,79	4|1|3 0,55	16|1|0 0,77	2|3|0 0,61	2|1|3 0,43	7|0|0 0,79	0|0|6 0,21	4|0|3 0,56	6|0|0 0,79	0|0|1 0,21	1|0|0 0,79
Extensibility	2|0|0 0,79	3|0|1 0,7	5|1|0 0,75	2|2|3 0,45	0|1|0 0,5	2|2|0 0,64	4|0|0 0,79	1|2|1 0,5	5|0|0 0,79	5|0|0 0,79	1|0|0 0,79	1|0|0 0,79
Security	1|0|0 0,79	2|0|1 0,65	0|0|7 0,21	9|1|0 0,76	1|2|0 0,58	4|0|1 0,72	5|0|1 0,73	0|0|6 0,21	1|0|0 0,79	1|0|0 0,79	4|0|0 0,79	1|0|1 0,5
Testability	0|0|2 0,21	0|0|2 0,21	6|1|1 0,71	7|0|1 0,75	0|0|4 0,21	0|2|3 0,32	0|1|3 0,27	0|0|8 0,21	3|1|0 0,73	1|1|0 0,64	3|0|1 0,7	0|0|1 0,21
Modifiability	1|0|0 0,79	2|0|1 0,65	5|0|2 0,67	6|1|1 0,71	3|0|1 0,7	0|0|3 0,21	2|0|0 0,79	5|0|1 0,73	2|0|0 0,79	1|0|0 0,79	1|0|0 0,79	
Flexibility	2|0|0 0,79	6|0|0 0,79	4|0|0 0,79	7|0|1 0,75		2|0|0 0,79	1|0|0 0,79	1|0|0 0,79	3|0|0 0,79	3|0|0 0,79	5|0|0 0,79	5|0|0 0,79
Reliability	2|1|0 0,7	5|0|1 0,73	1|1|7 0,28	7|0|1 0,75	0|1|3 0,27	1|1|3 0,35	1|1|1 0,5	0|2|3 0,32	0|1|0 0,5	6|0|0 0,79	3|0|0 0,79	1|1|0 0,64
Availability	1|0|0 0,79	3|0|0 0,79	2|0|4 0,36	4|2|0 0,7	1|2|1 0,5	1|1|3 0,35	2|0|0 0,79	0|0|4 0,21			1|0|0 0,79	3|0|0 0,79
Adaptability	2|0|0 0,79	2|0|0 0,79	0|1|0 0,5	2|0|0 0,79	1|1|0 0,64	0|1|0 0,5	2|0|0 0,79	0|0|1 0,21	2|0|0 0,79	1|0|0 0,79	3|0|1 0,7	1|0|0 0,79
Analyzability	1|0|0 0,79	1|0|1 0,5	2|1|0 0,7	1|0|1 0,5	0|1|0 0,5	1|0|1 0,5	1|0|0 0,79	0|1|1 0,36	0|1|0 0,5	1|0|0 0,79		
Complexity	0|0|2 0,21		0|1|1 0,36	1|0|2 0,35	0|2|0 0,5	0|1|1 0,36	0|0|1 0,21	2|0|0 0,79	2|1|1 0,6	2|0|1 0,65	3|0|2 0,59	1|0|0 0,79
Maintainability	4|1|2 0,61	2|0|0 0,79	5|2|0 0,71	10|1|1 0,74	1|1|0 0,64	0|1|0 0,5	4|0|0 0,79	4|3|0 0,67	2|0|3 0,41	3|1|0 0,73	3|0|0 0,79	2|0|0 0,79
Modularity	2|0|0 0,79	3|0|1 0,7	0|0|2 0,21	1|1|0 0,64	1|1|0 0,64	1|1|0 0,64	1|0|0 0,79	0|0|2 0,21	1|0|1 0,5	1|0|0 0,79	3|0|0 0,79	

Ejemplo: Layered

- Las componentes en una arquitectura de este tipo están organizadas en Layers horizontales, cada uno de ellos responsable de un rol específico en la aplicación (negocios, presentación, persistencia).
- Cada layer no conoce detalles del “siguiente” lo cual provee una forma simple de separación de concerns
- El número de layers no esta predeterminado pero la mayoría de las aplicaciones usan entre 3 y 5

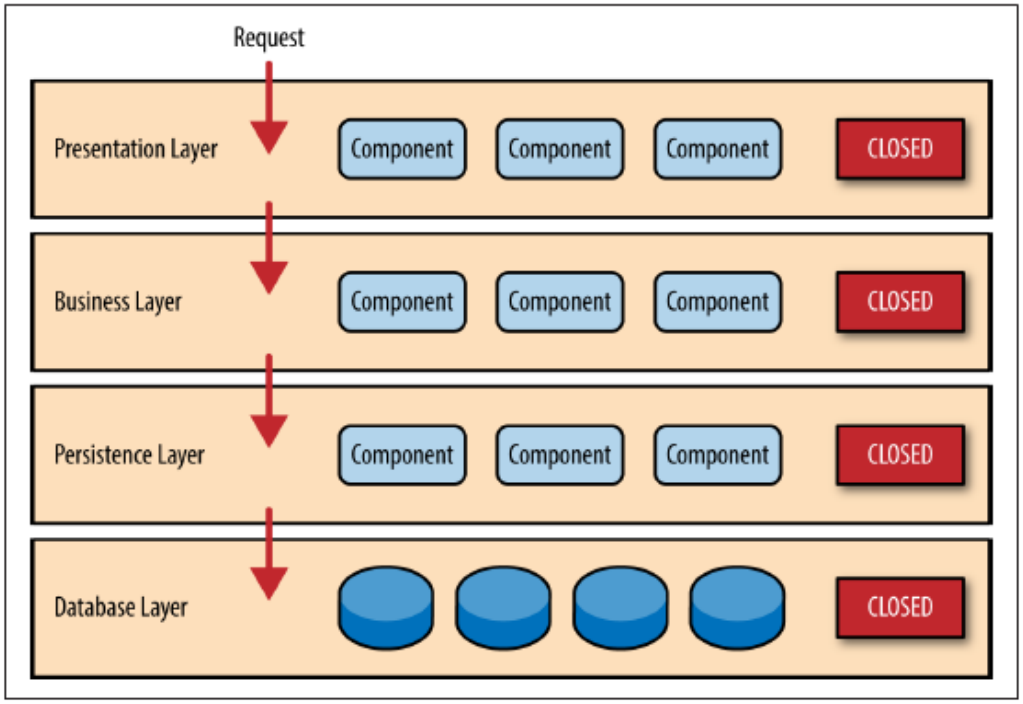
Ejemplo: Layered

- Las componentes en una arquitectura de este tipo están organizadas en Layers horizontales, cada uno de ellos responsable de un rol específico en la aplicación (negocios, presentación, persistencia).
- Cada layer no conoce detalles del "siguiente" lo cual provee una forma simple de separación de concerns
- El número de layers no esta predeterminado pero la mayoría de las aplicaciones usan entre 3 y 5



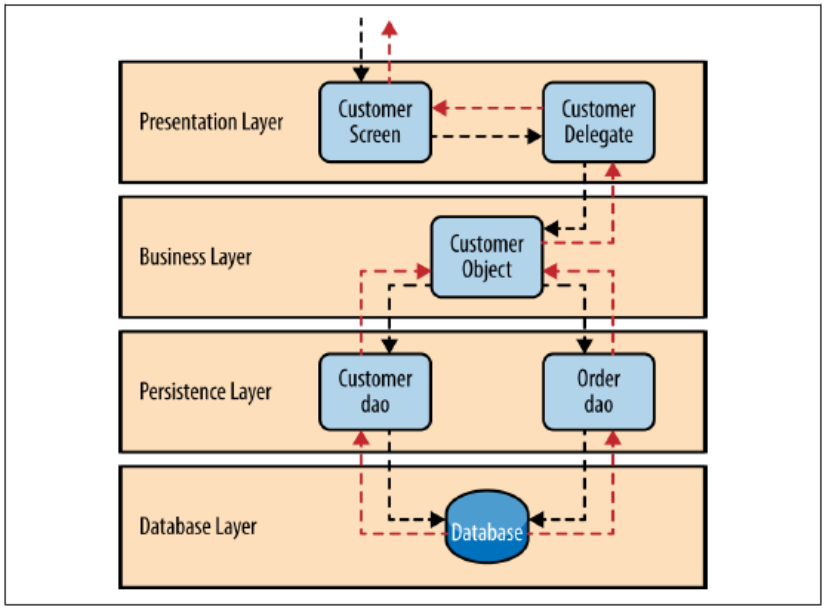
Conceptos claves

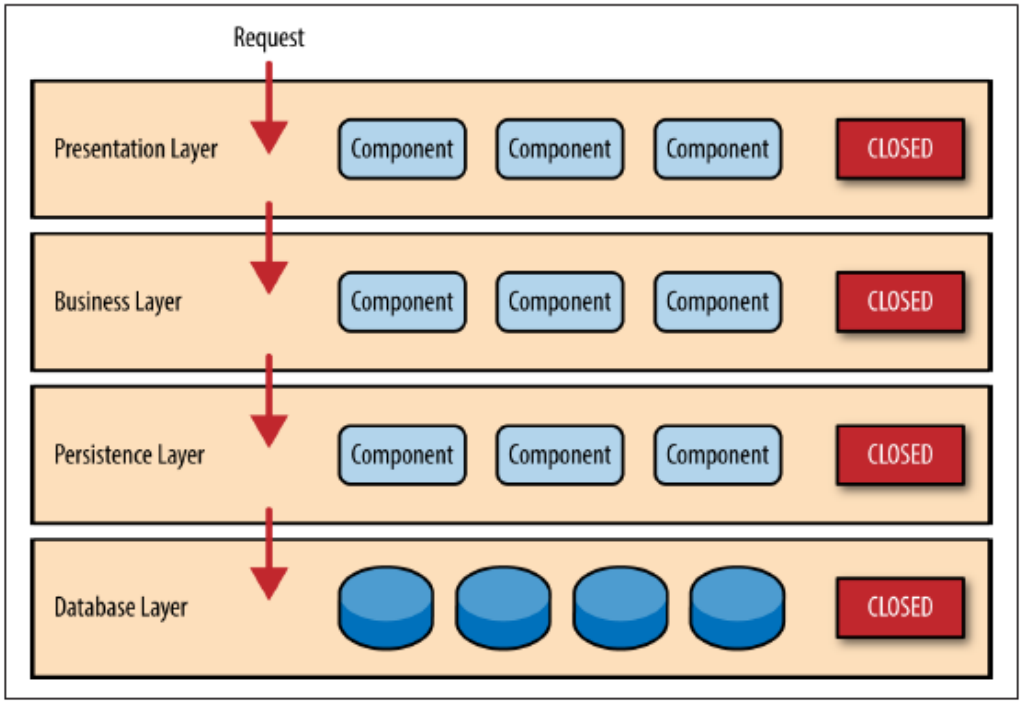
- En las arquitecturas layered se dice que los layers son "cerrados", todos los requests deben ser tratados por el layer inmediatamente inferior
- Esto permite aislar mejor los cambios y provee independencia entre layers



Conceptos claves

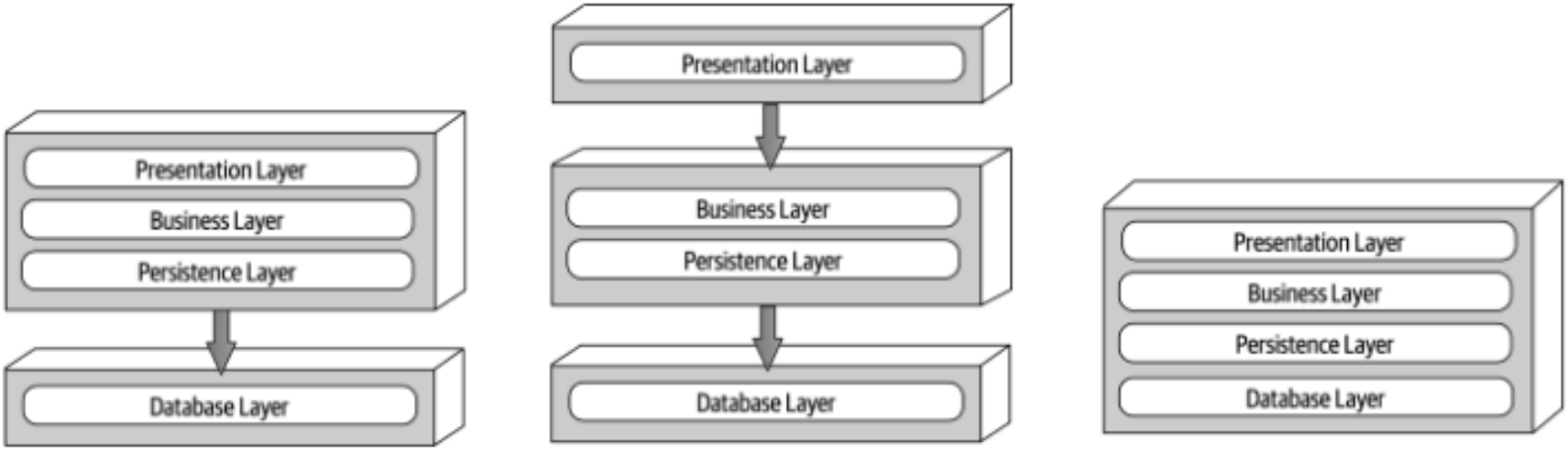
- En las arquitecturas layered se dice que los layers son "cerrados", todos los requests deben ser tratados por el layer inmediatamente inferior
- Esto permite aislar mejor los cambios y provee independencia entre layers





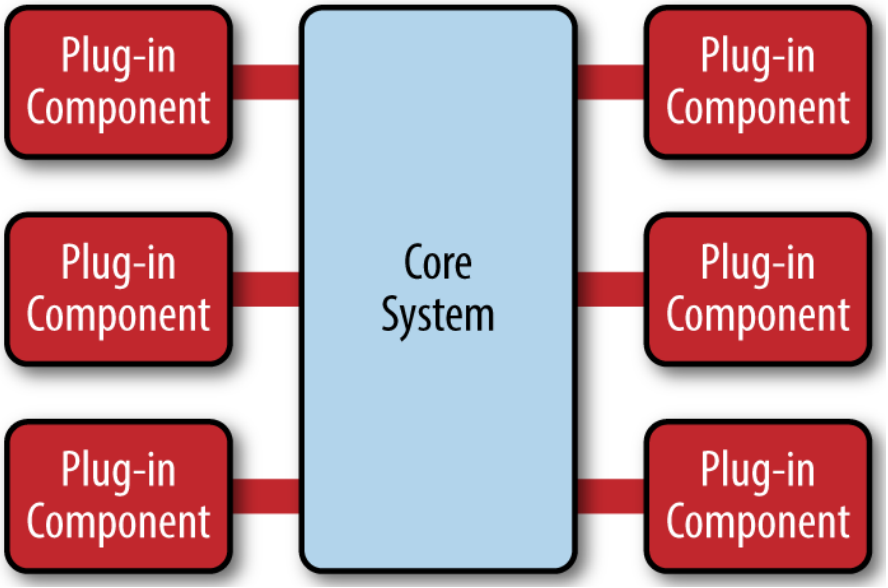
Deployment

- El modelo de Layers no presupone una decisión de asociación entre layers y hardware
- Es posible elegir deployments diferentes



Ejemplo: Microkernel

- En el estilo Microkernel tenemos dos tipos de componentes: El Core y componentes Plugins
- La Lógica de la aplicación se divide entre el sistema "básico" y los plugins y provee de esta manera flexibilidad, extensibilidad como consecuencia del desacople entre la funcionalidad "básica" y la lógica provista por los plugins



Microkernel

- Muy usado en arquitectura de “productos” (como por ejemplo Eclipse o Moodle) y Web Browsers
- La estructura original no postula de que manera el Core System se comunica con los plugins (solo que deben ser independientes)
- La comunicación puede darse incluso usando otros estilos arquitecturales (micro-servicios) o mediante call-backs a partir de un registro de plugins

Arquitectura vs. Diseño

- Toda arquitectura implica diseño pero no todo diseño implica arquitectura
- Una posible diferencia está en cuales decisiones son más significativas (por ejemplo en relación al costo de cambiarlas)
- Ejemplos de decisiones "significativas" de distinto tipo:
  - La "forma" del sistema: client-server, Web, nativo móvil, distribuido, asincrónico, etc
  - La estructura de software: componentes, capas, interacciones, etc
  - La elección de tecnología: lenguaje de programación, plataformas de deployment
  - La elección de frameworks de desarrollo: MVC, persistencia, etc

Arquitectura vs Diseño. Ejemplo

- Muchas veces la elección de una base de datos relacional es una decisión significativa (cual elegimos)
- Sin embargo si decidimos usar un framework de Object-Relational Mapping para desacoplar la base de datos, el tipo de BD dejo de ser significativo pero el layering agregado por el framework pasó a serlo.
- Elegimos desacoplar la BD agregando una capa de complejidad. Eliminamos una decisión significativa y agregamos una diferente

Problemas a enfrentar

- En este contexto tendremos que resolver problemas de tipo:
  - Micro: diseño (OO)
  - Micro-arquitectural: usando patrones
  - De más alto nivel: que arquitectura elegimos
  - Uso de frameworks (básicos tipo UI, avanzados tipo Cloud o persistencia)