Universidad Rafael Landívar
Tradición Jesuita en Guatemala

Análisis y Diseño II

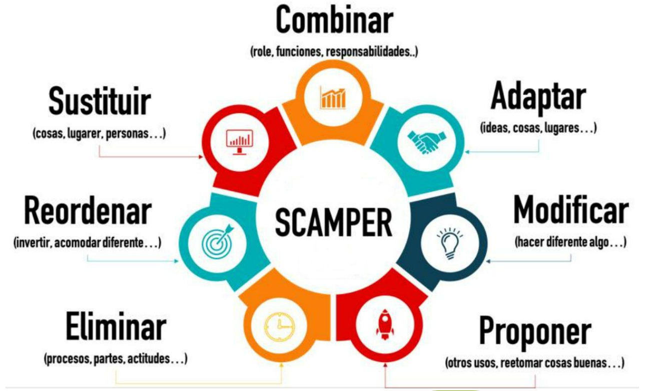
Arquitectura por capas

Agenda

1. Bienvenida y ejercicio de creatividad.
2. Feedback micro-enseñanzas
3. RAP.
4. Aplicación.
5. Pasos siguientes.

Siga los siguientes pasos:

En equipos como están sentados.


Combinar
(role, funciones, responsabilidades..)
Sustituir
(cosas, lugarer, personas…)
Adaptar
(ideas, cosas, lugares…)
Reordenar
(invertir, acomodar diferente…)
SCAMPER
Modificar
(hacer diferente algo…)
Eliminar
(procesos, partes, actitudes…)
Proponer
(otros usos, reetomar cosas buenas…)

Paso 1

Elija un objeto cotidiano

Paso 2

¿Cuál sería la nueva versión del objeto según SCAMPER? Utilice solo una letra del acrónimo.

Paso 3

Describa la nueva versión del objeto.

Feedback

Micro-enseñanzas primer grupo

Feedback - Micro-enseñanzas

¿Qué recomendaciones le doy al equipo y a futuros equipos?

¿Qué preocupaciones me generó la actividad?

¿Qué valoro de lo que hizo el equipo?

¿Qué dudas me quedan?

iRAT

Clave: monolitica

gRAT

Clave: ncapas

Apelaciones

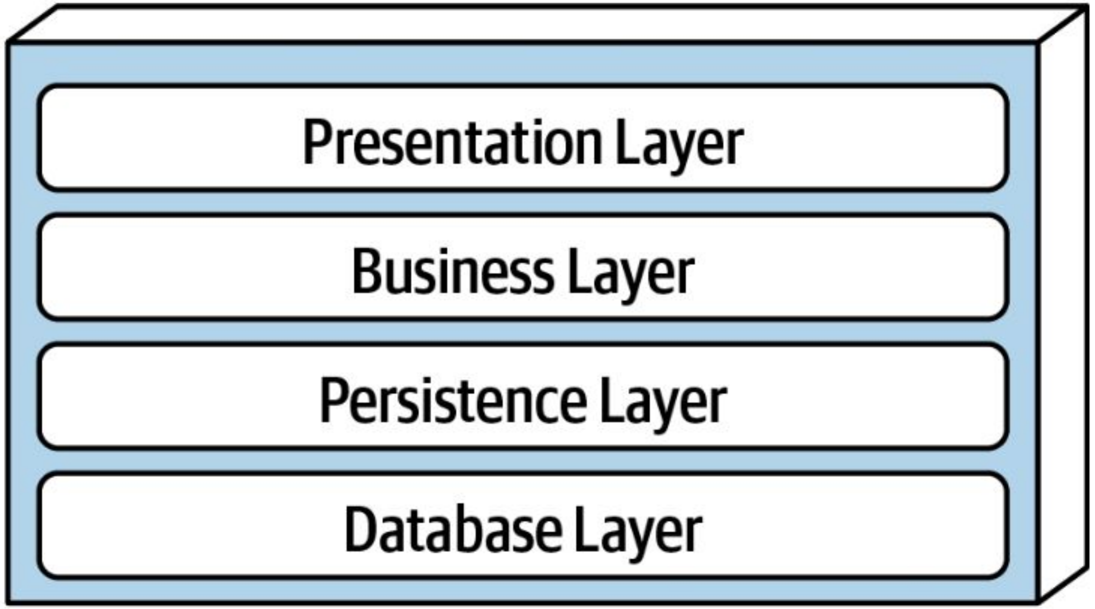


Figure 10-1. Standard logical layers within the layered architecture style

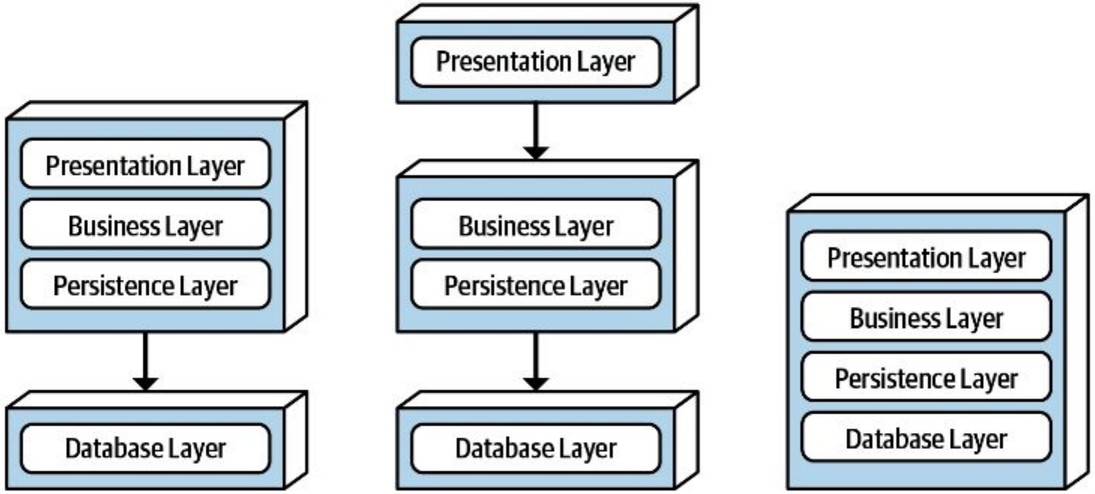


Figure 10-2. Physical topology (deployment) variants

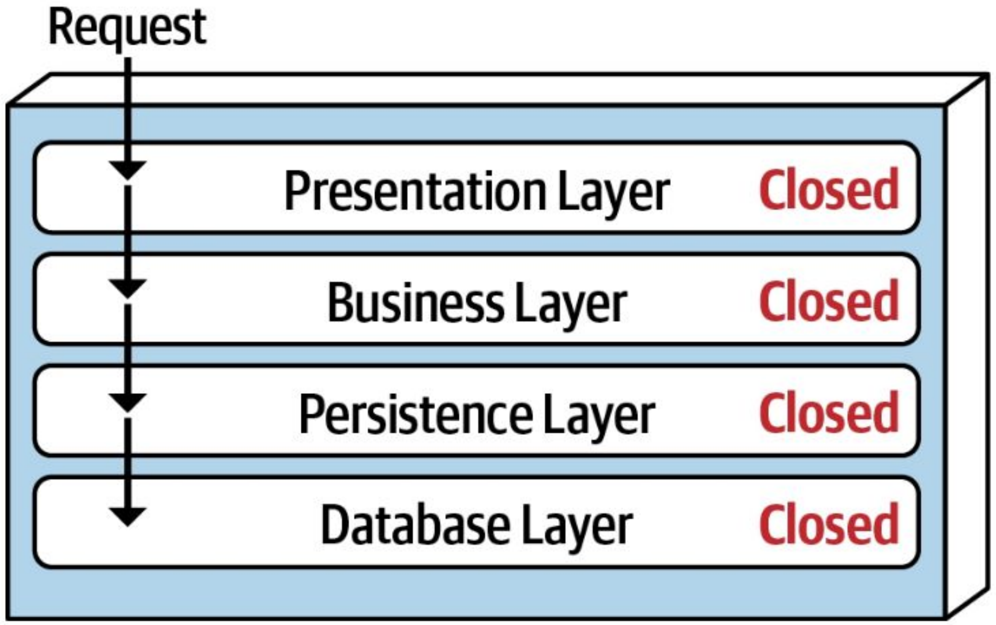


Figure 10-3. Closed layers within the layered architecture

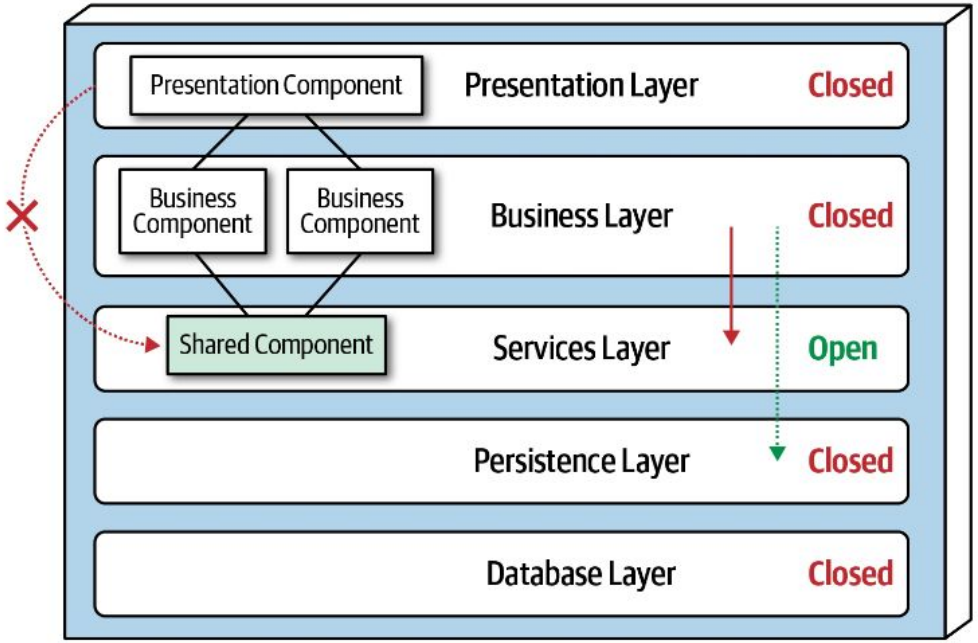
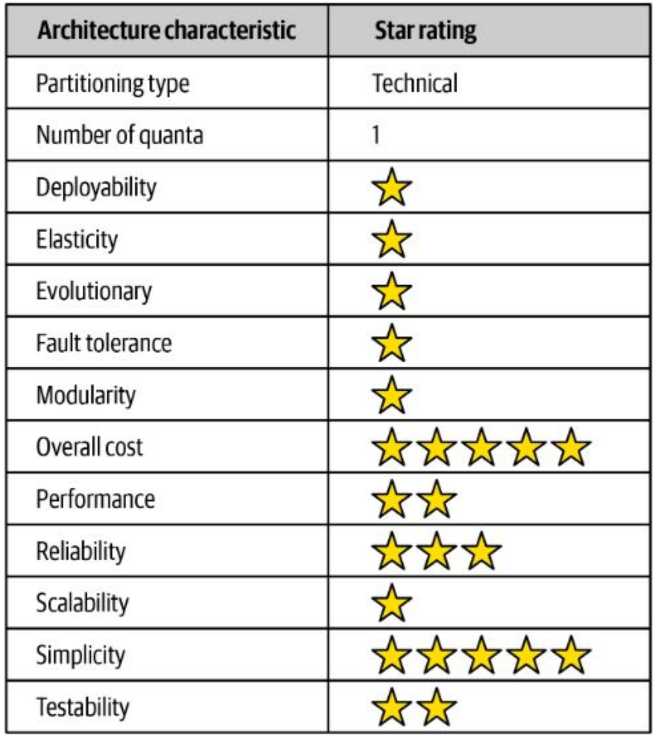
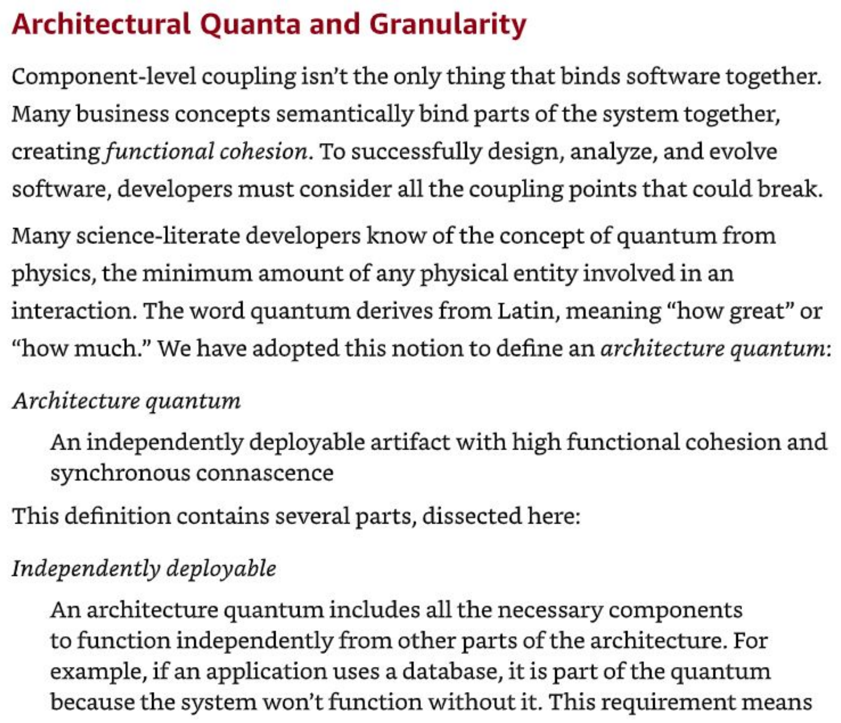


Figure 10-5. Adding a new services layer to the architecture



Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★
Elasticity	★
Evolutionary	★
Fault tolerance	★
Modularity	★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★



Architectural Quanta and Granularity

Component-level coupling isn't the only thing that binds software together. Many business concepts semantically bind parts of the system together, creating *functional cohesion*. To successfully design, analyze, and evolve software, developers must consider all the coupling points that could break.

Many science-literate developers know of the concept of quantum from physics, the minimum amount of any physical entity involved in an interaction. The word quantum derives from Latin, meaning "how great" or "how much." We have adopted this notion to define an *architecture quantum*:

Architecture quantum

An independently deployable artifact with high functional cohesion and synchronous connascence

This definition contains several parts, dissected here:

Independently deployable

An architecture quantum includes all the necessary components to function independently from other parts of the architecture. For example, if an application uses a database, it is part of the quantum because the system won't function without it. This requirement means that virtually all legacy systems deployed using a single database by definition form a quantum of one. However, in the microservices architecture style, each service includes its own database (part of the *bounded context* driving philosophy in microservices, described in detail in Chapter 17), creating multiple quanta within that architecture.

High functional cohesion

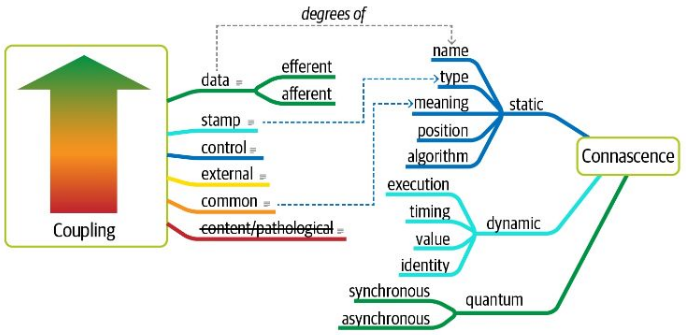
Cohesion in component design refers to how well the contained code is unified in purpose. For example, a `Customer` component with properties and methods all pertaining to a *Customer* entity exhibits high cohesion; whereas a `Utility` component with a random collection of miscellaneous methods would not.

High functional cohesion implies that an architecture quantum does something purposeful. This distinction matters little in traditional monolithic applications with a single database. However, in microservices architectures, developers typically design each service to match a single workflow (a *bounded context*, as described in "Domain-Driven Design's Bounded Context"), thus exhibiting high functional cohesion.

Synchronous connascence

Synchronous connascence implies synchronous calls within an application context or between distributed services that form this architecture quantum. For example, if one service in a microservices architecture calls another one synchronously, each service cannot exhibit extreme differences in operational architecture characteristics. If the caller is much more scalable than the callee, timeouts and other reliability concerns will occur. Thus, synchronous calls create dynamic connascence for the length of the call — if one is waiting for the other, their operational architecture characteristics must be the same for the duration of the call.

Back in Chapter 6, we defined the relationship between traditional coupling metrics and connascence, which didn't include our new *communication connascence* measure. We update this diagram in Figure 7-1.



2

Aplicación

Distribuyendo los componentes en capas

Componentes en capas

Con base al ejercicio de Katas y componentes.
Modelar los componentes por capas.

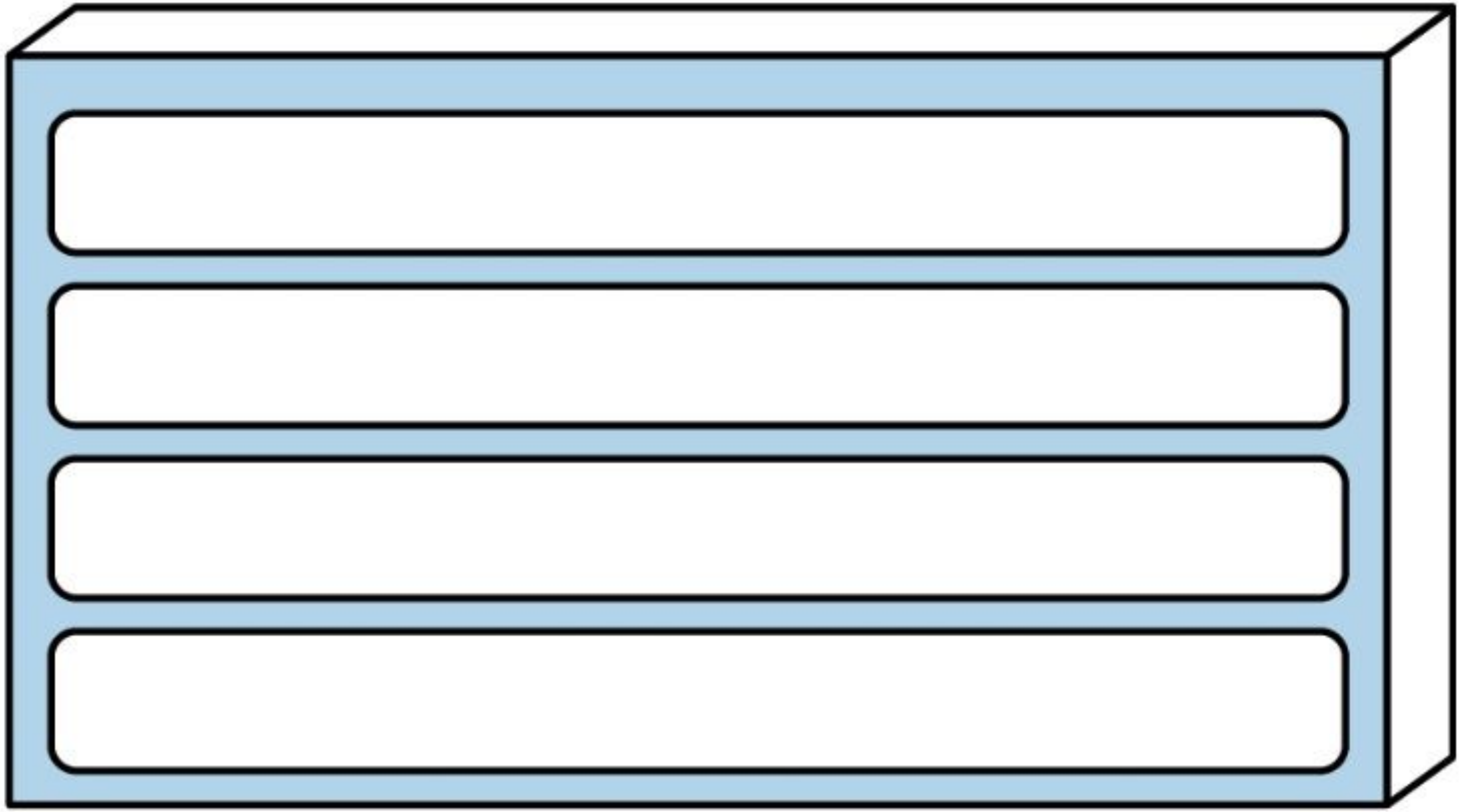


Figure 10-1. Standard logical layers within the layered architecture style

Distribuya su diseño de componentes

Pasos siguientes

- Primer examen parcial: 8-junio-2023.
- Preparación para e el laboratorio.
- Google Cloud Developer.

Pase de salida sobre la clase:

¿Qué fue lo que más me gustó y aprendí?	¿Qué fue lo que no me gustó o no aprendí?
¿Qué se me facilitó?	¿Qué se me dificultó?

Créditos

Universidad Rafael Landívar

Tradición Jesuita en Guatemala