



Luca Cabibbo
Architettura
dei Sistemi
Software

Architettura esagonale

dispensa asw360
ottobre 2024

*There must be a cause why snowflakes
have the shape of six-cornered starlets.
It cannot be chance. Why always six?*

Johannes Kepler

1

Architettura esagonale

Luca Cabibbo ASW



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 20, **Architettura esagonale**
- ❑ Cockburn, A. **Hexagonal Architecture**. 2005.
 - <https://alistair.cockburn.us/hexagonal-architecture/>
- ❑ Cockburn, A. and Garrido de Paz, J.M. **Hexagonal Architecture Explained: How the Ports & Adapters architecture simplifies your life, and how to implement it**. Humans and Technology Press, v0.9b. 2024.
- ❑ Vernon, V. **Implementing Domain-Driven Design**. Addison-Wesley, 2013.
- ❑ Richardson, C. **Microservices Patterns: With examples in Java**. Manning, 2019.

2

Architettura esagonale

Luca Cabibbo ASW



- Obiettivi e argomenti

- Obiettivi
 - presentare l'architettura esagonale

- Argomenti
 - introduzione
 - architettura a strati e inversione delle dipendenze
 - architettura esagonale
 - discussione



* Introduzione

- L'architettura esagonale è un pattern architetturale comune
 - è diffuso soprattutto nel contesto di DDD e dell'architettura a microservizi
 - in un sistema distribuito, consente di definire l'architettura di un singolo componente o servizio software, nonché di gestire le interazioni di questi componenti o servizi



Responsabilità di business e infrastrutturali

- Responsabilità degli elementi software
 - *responsabilità di business* – o *logica di business*
 - una o più funzionalità applicative
 - *responsabilità infrastrutturali*
 - sono relative all'accesso ai servizi infrastrutturali
 - un *servizio infrastrutturale* è un servizio tecnico (non applicativo) – ad es., un database o un servizio di comunicazione distribuita
 - dipendono da tecnologie specifiche e richiedono l'utilizzo di framework tecnici opportuni



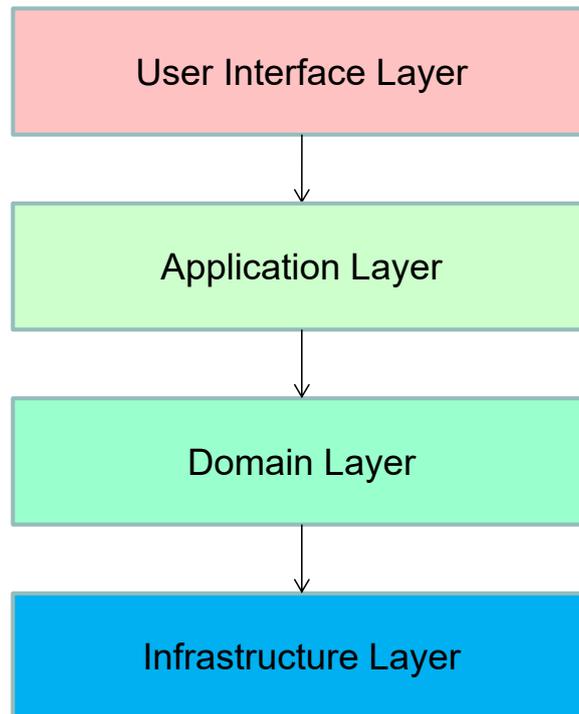
Responsabilità di business e infrastrutturali

- Come allocare le responsabilità di business e infrastrutturali agli elementi software? Intuitivamente
 - le responsabilità di business competono ai “componenti”
 - le responsabilità infrastrutturali competono ai “connettori”
 - le responsabilità di business dovrebbero essere disaccoppiate (non dovrebbero dipendere) da quelle infrastrutturali



* Architettura a strati e inversione delle dipendenze

- Consideriamo di nuovo l'architettura a strati "tradizionale" – ad es., la *Layered Architecture* di DDD



7

Architettura esagonale

Luca Cabibbo ASW



DIP e architettura a strati

- Consideriamo di nuovo anche il *principio di inversione delle dipendenze (DIP)*
 - i moduli di alto livello (più importanti) non dovrebbero dipendere dai moduli di basso livello (meno importanti) – piuttosto, entrambi dovrebbero dipendere da opportune astrazioni
 - le astrazioni non dovrebbero dipendere dai dettagli – piuttosto, i dettagli dovrebbero dipendere dalle astrazioni
 - l'architettura a strati tradizionale soddisfa il DIP?

8

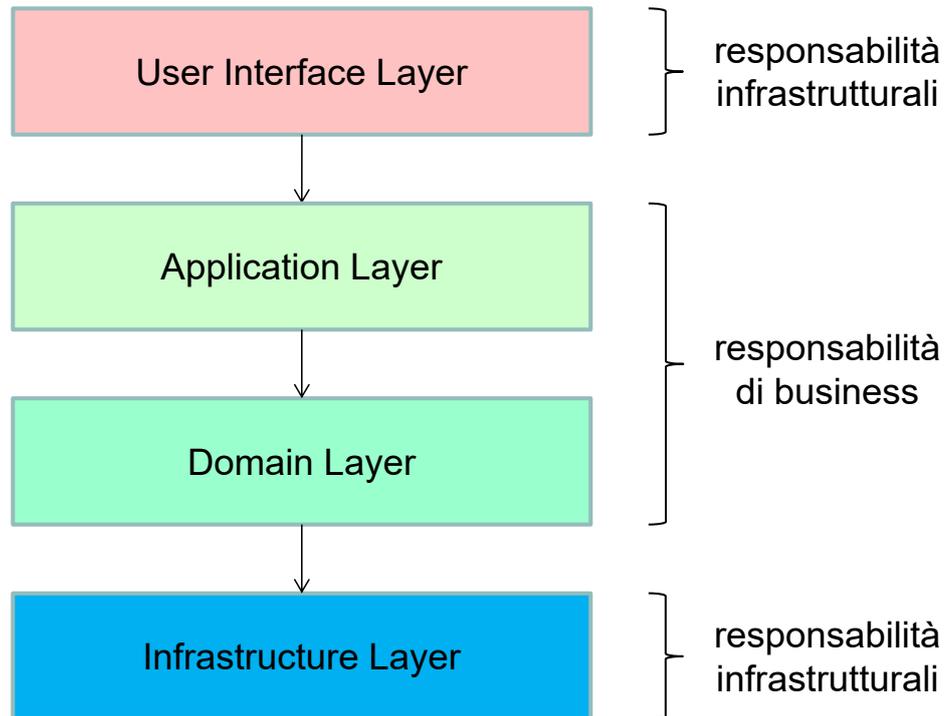
Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP

- L'architettura a strati tradizionale soddisfa il DIP?



9

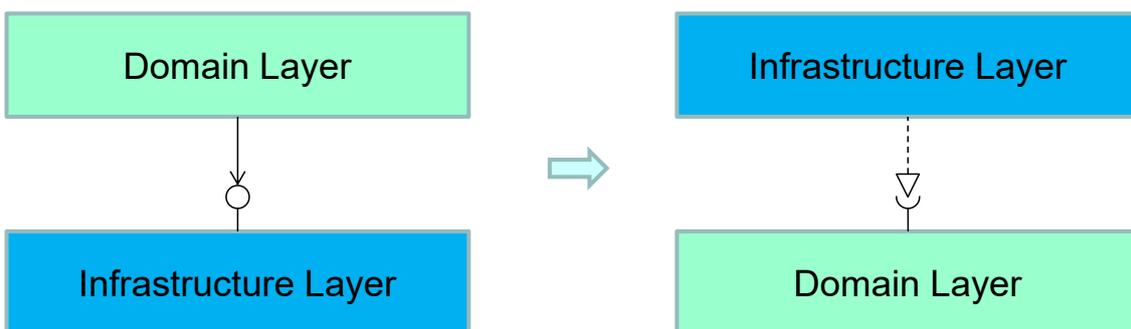
Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP

- L'architettura a strati tradizionale non soddisfa il DIP
 - è però possibile applicare il DIP, introducendo delle “opportune astrazioni”



- l'idea è definire delle interfacce per le responsabilità infrastrutturali negli strati di business in cui queste responsabilità sono richieste e utilizzate – e implementarle nello strato dell'infrastruttura
 - ad es., per l'accesso ai dati persistenti, oppure per l'invio di messaggi su un canale per messaggi

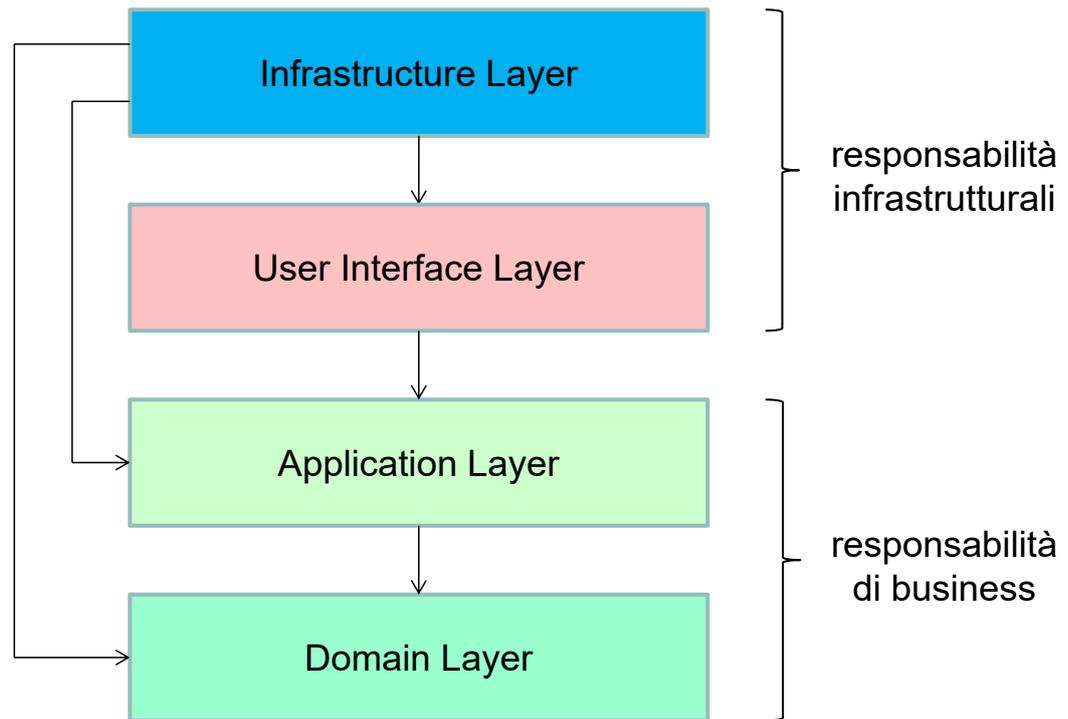
10

Architettura esagonale

Luca Cabibbo ASW



Architettura a strati e DIP



11

Architettura esagonale

Luca Cabibbo ASW



Verso l'architettura esagonale

- L'architettura esagonale, intuitivamente
 - ha origine nell'architettura a strati
 - a cui però viene prima applicato il DIP
 - e poi viene rimossa l'asimmetria sopra-sotto dell'architettura a strati

12

Architettura esagonale

Luca Cabibbo ASW



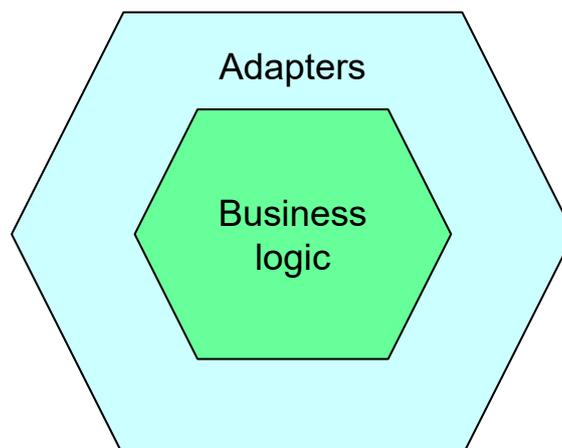
* Architettura esagonale

- Il pattern architetturale **Hexagonal Architecture** (*architettura esagonale*) – chiamato anche, in modo più descrittivo, **Ports and Adapters**
 - nel contesto di un sistema software distribuito, aiuta a organizzare un singolo componente distribuito – che chiamiamo un *servizio (servizio applicativo) o componente esteso*
 - in pratica, ogni servizio ha sia responsabilità di business che responsabilità di presentazione e infrastrutturali
 - questo pattern sostiene
 - un accoppiamento debole tra le responsabilità di business e le altre responsabilità
 - un'interazione flessibile tra i servizi – tra di loro e con altre entità esterne



Architettura esagonale

- Ogni servizio è rappresentato da un “esagono”
 - ogni servizio/componente esteso/esagono è inoltre suddiviso in due parti
 - *interno (inside)* – responsabilità di business
 - *esterno (outside)* – responsabilità di presentazione e infrastrutturali





Architettura esagonale

- L'*interno* di un servizio (*logica di business*) – chiamato anche *application*, *app* o *core* – implementa un insieme di funzionalità e responsabilità di business
 - implementa gli strati Domain e Application del servizio applicativo
 - non implementa responsabilità di presentazione o infrastrutturali
 - tuttavia le supporta, definendo delle interfacce utili per la presentazione e per l'accesso ai servizi infrastrutturali
 - in pratica, l'interno è la parte del servizio attorno a cui vogliamo mettere dei confini
 - dentro c'è tutta e sola la logica di business
 - fuori c'è tutta e sola la tecnologia legata all'uso dei servizi infrastrutturali



Architettura esagonale

- Il servizio (la sua logica di business) ha la necessità di interagire con diversi tipi di *entità esterne* – chiamati anche *attori*
 - ad es., gli utenti, i test automatizzati e la base di dati, oppure anche altri servizi o servizi di altre organizzazioni
 - si vuole che la logica di business possa interagire con altre entità esterne
 - con un accoppiamento basso – evitando l'intreccio tra la logica di business e le interazioni con le entità esterne
 - in modo flessibile



Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori
 - ciascuna *porta* (definita nella logica di business) definisce un'interfaccia (fornita o richiesta) che rappresenta una modalità di interazione con la logica di business, con uno scopo specifico
 - due tipi di porte
 - *inbound port* – un'interfaccia fornita
 - è chiamata anche una *primary port* – perché consente ad un attore primario esterno di usare le funzionalità della logica di business
 - *outbound port* – un'interfaccia richiesta
 - è chiamata anche una *secondary port* – perché consente alla logica di business di usare le funzionalità di attore di supporto (secondario) esterno



Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori
 - esempi di porte per un servizio S
 - porte per interagire con i suoi utenti (sul web) e per ricevere chiamate remote da altri servizi
 - porte per accedere alla base di dati e per effettuare chiamate remote ad altri servizi
 - porte per lo scambio asincrono di messaggi



Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori
 - intuitivamente, gli adattatori implementano lo strato Presentation e lo strato Infrastructure del servizio
 - ciascun *adattatore* (definito nell'esterno del servizio) è relativo a una specifica porta, ed ha lo scopo di adattare le interazioni tra quella porta e un'entità esterna, utilizzando una tecnologia specifica
 - due tipi di adattatori – a seconda del tipo di porta a cui si riferiscono
 - *inbound adapter* – chiamato anche *primary adapter*
 - *outbound adapter* – chiamato anche *secondary adapter*

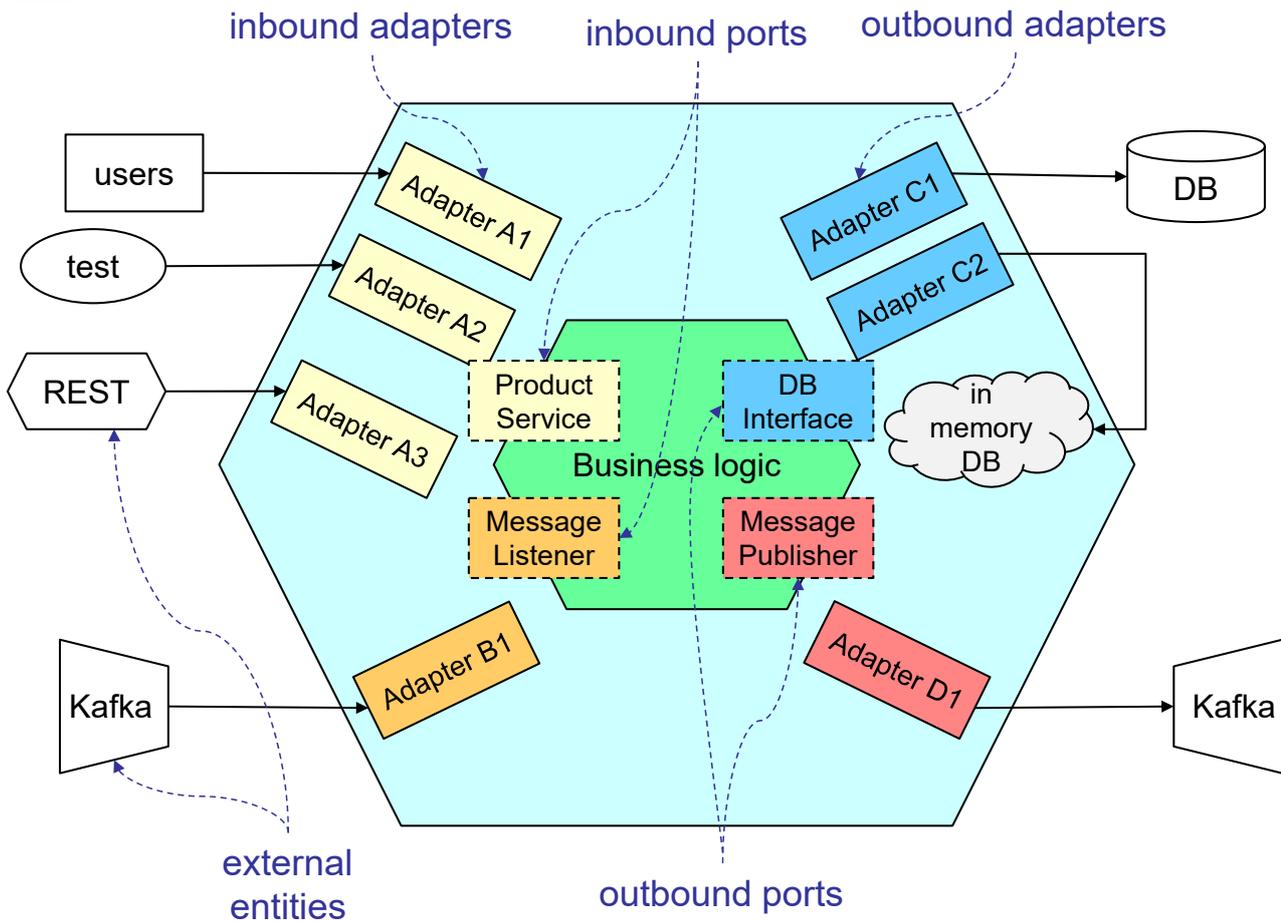


Porte e adattatori

- La logica di business interagisce con le entità esterne tramite porte e adattatori
 - esempi di adattatori per un servizio S
 - un controller Spring Web MVC
 - un adattatore per l'accesso a una base di dati MySQL – un altro per l'accesso a una base di dati inmemory
 - adattatori per ricevere ed effettuare chiamate REST – e altri per chiamate gRPC
 - adattatori per inviare e ricevere messaggi con Kafka



Architettura esagonale



21

Architettura esagonale

Luca Cabibbo ASW



Porte

- Una **porta** rappresenta una modalità di interazione con il servizio (con la sua logica di business), con uno scopo specifico
 - una nozione volutamente generica e flessibile
 - le porte sono rappresentate graficamente dai lati dell'esagono
 - ma non vuol dire che ogni servizio debba avere sei porte
 - ogni porta è in corrispondenza con un'interfaccia interna (un'API) della logica di business
 - ad es., in DDD potrebbero essere dei service, repository e application service

22

Architettura esagonale

Luca Cabibbo ASW



Adattatori

- Ciascun *adattatore* è relativo all'interazione con una specifica tipologia di entità esterna – adatta il tipo di interazione richiesto dall'entità esterna all'API interna dell'applicazione, e viceversa, utilizzando una tecnologia specifica
 - ad esempio
 - un controller web MVC, l'implementazione di un repository JPA, un controller REST, un endpoint per messaggi Kafka
 - per ciascuna porta ci possono essere più adattatori
 - ciascun adattatore può essere utilizzato anche da più entità esterne differenti



Scenari

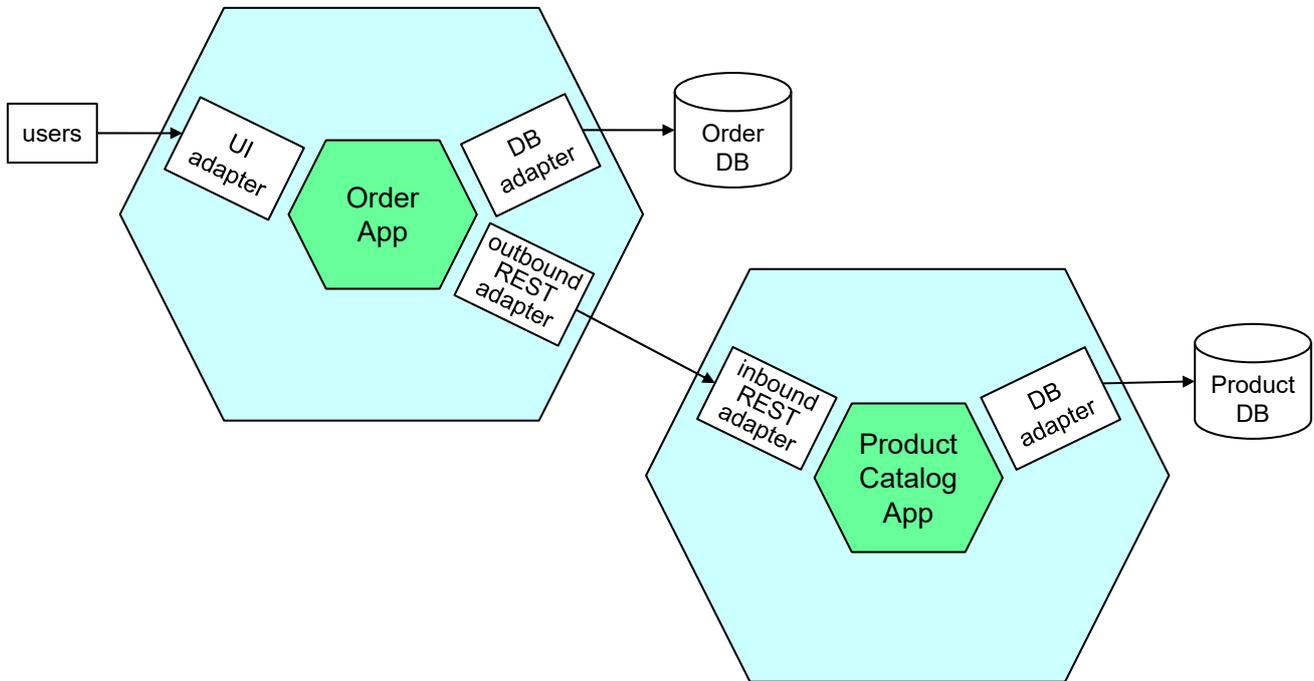
- Alcuni scenari principali
 - un'entità esterna interagisce con un servizio attraverso un adattatore specifico, relativo a una porta specifica
 - l'adattatore converte questo evento esterno in un'invocazione di un'operazione o in un messaggio, e lo passa alla logica di business tramite l'interfaccia di quella porta (inbound)
 - la logica di business di un servizio deve interagire con l'esterno
 - la logica di business interagisce con l'interfaccia di una porta (outbound) – l'adattatore associato a quella porta converte la richiesta o il messaggio in un formato appropriato per l'entità esterna
 - in entrambi i casi, la logica di business è indipendente sia dalla natura dell'entità esterna che dagli specifici adattatori utilizzati e dalle tecnologie sottostanti



Scenari

Alcuni scenari principali

- un sistema software distribuito composto da più servizi, che interagiscono tra di loro mediante porte e adattatori



25

Architettura esagonale

Luca Cabibbo ASW



Conseguenze

L'architettura esagonale ha impatto positivo su diverse qualità

- modificabilità e flessibilità

- ☺ l'accoppiamento debole tra l'interno e l'esterno di un servizio sostiene la possibilità di implementare e di far evolvere la logica di business in modo indipendente dagli adattatori
- ☺ l'indirezione fornita dagli adattatori consente un'interazione flessibile di un servizio con altre entità esterne e con altri servizi
- ☺ l'architettura esagonale è compatibile con team cross-funzionali

26

Architettura esagonale

Luca Cabibbo ASW



Conseguenze

- L'architettura esagonale ha impatto positivo su diverse qualità
 - verificabilità
 - 😊 l'ampio uso di interfacce sostiene l'utilizzo di test double
 - 😊 è possibile verificare separatamente la logica di business e gli adattatori
 - 😊 è possibile usare adattatori specifici per i test (ad es., per un inmemory database)
 - interoperabilità
 - 😊 grazie alle indirezioni fornite dagli adattatori
 - prestazioni
 - 😞 possono essere penalizzate dall'uso degli adattatori



Architettura esagonale e Layers

- Abbiamo detto che l'architettura esagonale ha origine nell'architettura a strati
 - ogni servizio è internamente basato su due "strati"
 - sostiene un isolamento delle responsabilità di business dalle responsabilità di presentazione e infrastrutturali
 - c'è però un'importante differenza con l'architettura a strati
 - nell'architettura a strati, gli strati possono essere allocati a team di sviluppo separati (team mono-funzionali)
 - nell'architettura esagonale, sono i servizi applicativi che vengono allocati a team di sviluppo separati (team cross-funzionali, che si occupano di interi servizi)



Architettura esagonale, servizi, componenti e connettori

- Nell'architettura esagonale, in un servizio
 - la logica di business (interno) ha responsabilità funzionali – da “componente”
 - gli adattatori (esterno) hanno responsabilità infrastrutturali – da “connettori”
 - viene dunque ancora effettuata una decomposizione tra componenti e connettori
 - si tratta però di una decomposizione di secondo livello – perché la decomposizione di primo livello è quella relativa ai servizi
 - un intero servizio viene anche chiamato “componente esteso” – perché comprende la sua logica di business (“componente”) insieme agli adattatori, che “estendono” il “componente” per renderlo un elemento software distribuito autonomo



Un ulteriore elemento – il configuratore

- Nell'architettura esagonale, c'è anche bisogno di un *configuratore* per connettere tutti gli elementi di un servizio – per connettere la logica di business con gli adattatori, tramite le porte
 - questo configuratore deve in genere
 - istanziare la logica di business
 - istanziare gli adattatori utilizzati
 - collegare ogni adattatore inbound con la corrispondente porta inbound
 - collegare ogni porta outbound con il corrispondente adattatore outbound
 - ad esempio, come configuratore è possibile usare un framework per l'iniezione delle dipendenze come Spring



- Architettura esagonale nelle esercitazioni

- Ciascun servizio applicativo (ad es., **restaurant-service**) viene strutturato con l'architettura esagonale, usando i seguenti package
 - un package di base del servizio – ad es., **asw.efood.restaurant-service**
 - un package che definisce l'interno (logica di business) del servizio, comprese tutte le sue porte – ad es., **asw.efood.restaurant-service.domain**
 - contiene le entità, i service e i repository (le interfacce) – service e repository sono porte
 - un package separato per ciascun adattatore – ad es.,
 - **asw.efood.restaurant-service.rest** definisce l'adattatore REST per ricevere invocazioni remote da altri servizi
 - **asw.efood.restaurant-service.accountingclient.rest** definisce l'adattatore REST per effettuare invocazioni remote al servizio **accounting-service** tramite REST

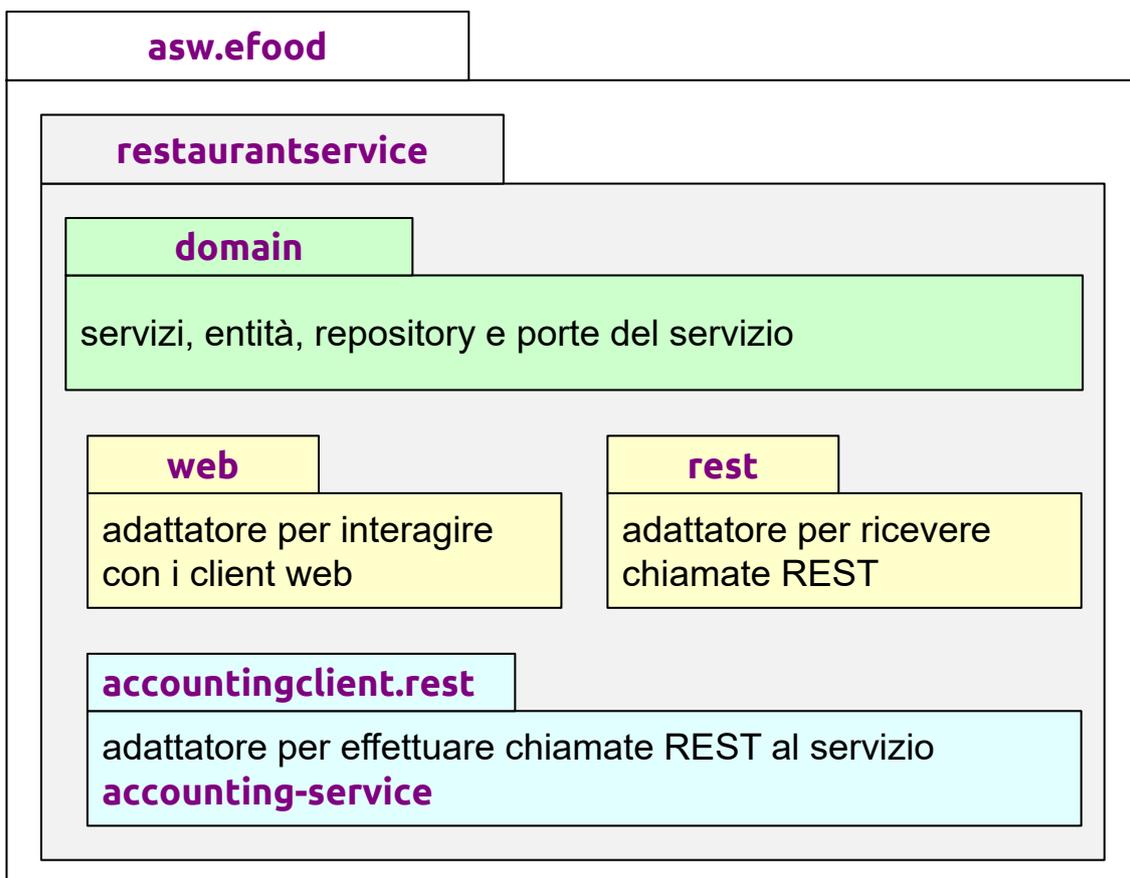
31

Architettura esagonale

Luca Cabibbo ASW



Architettura esagonale nelle esercitazioni



32

Architettura esagonale

Luca Cabibbo ASW



* Discussione

▣ L'architettura esagonale

- nell'ambito di un servizio, sostiene un isolamento delle responsabilità di business da quelle di presentazione e infrastrutturali
- può essere applicata nella realizzazione di sistemi distribuiti – ad es., nei microservizi
- è compatibile con DDD – ogni esagono può essere utilizzato per implementare un Bounded Context
- è un'architettura flessibile, compatibile con altri pattern – come l'architettura a servizi, i microservizi, gli eventi di dominio e CQRS