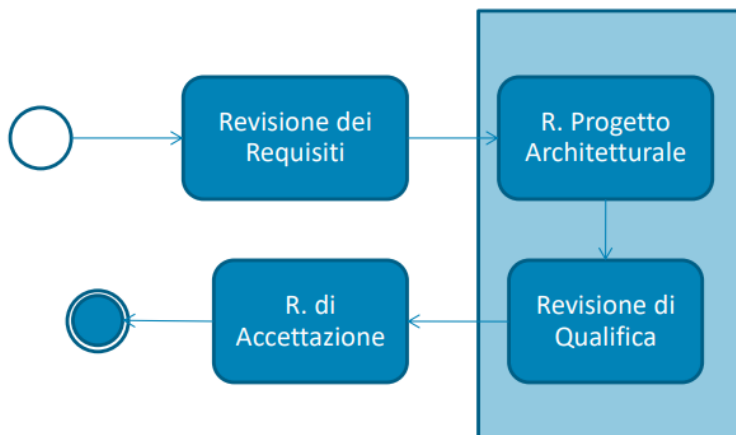


Sommario:

- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

1) Introduzione:



Progettare software object oriented è difficile:

Riusabilità:

- ➔ Ci vuole molta esperienza.

Soluzioni comuni e ricorrenti (pattern):

- ➔ Flessibili, eleganti, riusabili.

Design Pattern:

- ➔ Raccolta dell'esperienza dei progettisti, presentata in un modo effettivamente utilizzabile.
- ➔ Riutilizzo di soluzioni architetture vincenti

2) Cos'è un Design Pattern?

Ogni modello descrive un problema **che si verifica ripetutamente nel nostro ambiente**, e quindi descrive il nucleo della soluzione a quel problema, in modo tale da poter utilizzare questa soluzione un milione di volte, senza mai farlo due volte nello stesso modo.

Quattro elementi essenziali:

- ➔ Nome del pattern:
 - Vocabolario di progettazione.
- ➔ Il problema che il pattern risolve:
 - Descrizione del contesto.
- ➔ La soluzione:
 - Elementi, relazioni, responsabilità e collaborazioni
- ➔ Le conseguenze:

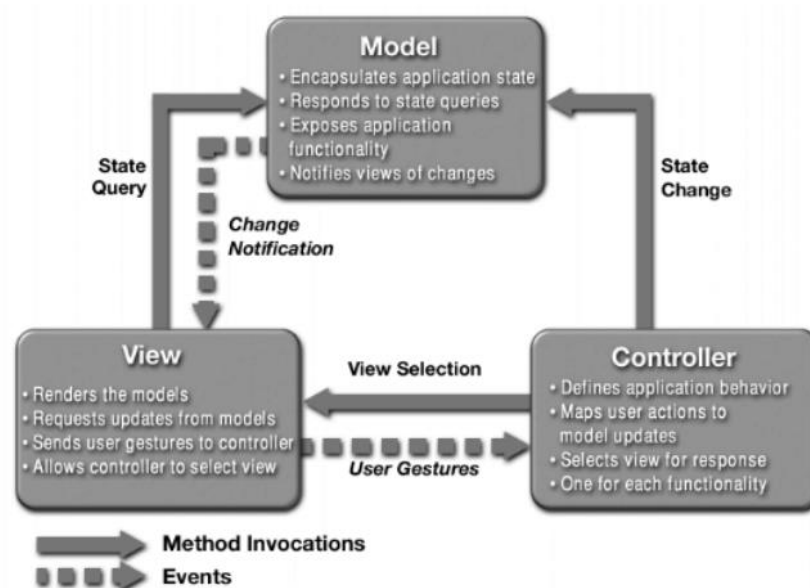
- Risultati e limiti della soluzione

ESEMPI DI DESIGN PATTERN:

Model-View-Controller (MVC):

Disaccoppia le tre componenti, rendendole riusabili.

- **Model:** dati di business e regole di accesso.
- **View:** rappresentazione grafica.
- **Controller:** reazione della UI agli input utente.

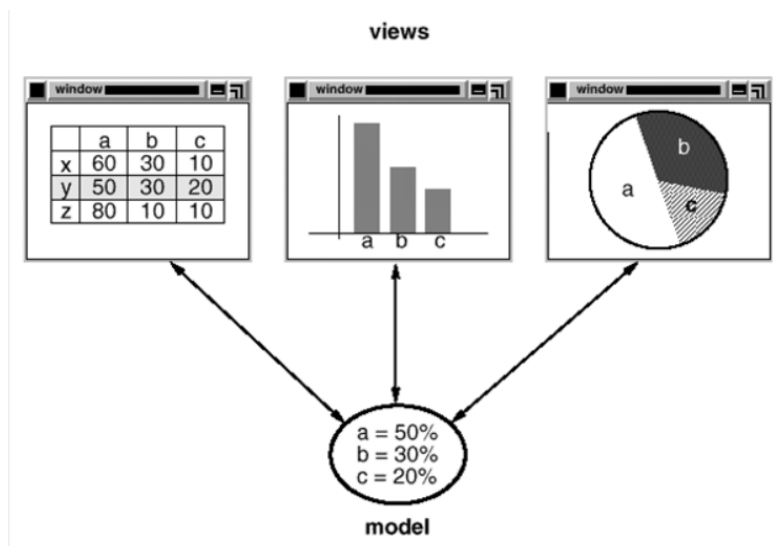


MVC: Model – View:

- Disaccoppiamento → protocollo di notifica.
- View deve garantire una visualizzazione consistente.

Diversi design pattern:

- Observer design pattern.
- Composite design pattern.



MVC: View – Controller:


Permette la modifica del comportamento in risposta all'input.

Strategy/Algorithm design pattern.

Descrizione di un Design Pattern:

- Convenzioni per la specifica
- Nome e classificazione
- Sinonimi: altri nomi noti del pattern
- Motivazione: problema progettuale
- Applicabilità: contesti in cui il pattern può essere applicato
- Struttura: rappresentazione grafica delle classi
- Partecipanti: classi e/o oggetti partecipanti e responsabilità
- Collaborazioni tra i partecipanti
- Conseguenze: costi e benefici
- Implementazione: suggerimenti, tecniche, errori comuni
- Esempio di codice sorgente: possibile implementazione
- Utilizzo comune: il pattern nei sistemi reali
- Pattern correlati

Classificazione Design Pattern

Relazioni tra		Campo di applicazione		
		Creational (5)	Structural (7)	Behavioral (11)
	Class	Factory method	Adapter (Class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter(Object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

3) Perché i design pattern?

Risolvere Problemi Architettureali:

- 1) Utilizzare gli oggetti **appropriati**.

È difficile decomporre il sistema in oggetti!

➔ Oggetti “reali” ≠ oggetti “non reali”.

DP permettono di identificare le astrazioni meno ovvie.

- 2) Utilizzare la giusta granularità:

Come decidere cosa deve essere un oggetto?

DP forniscono la granularità più appropriata.

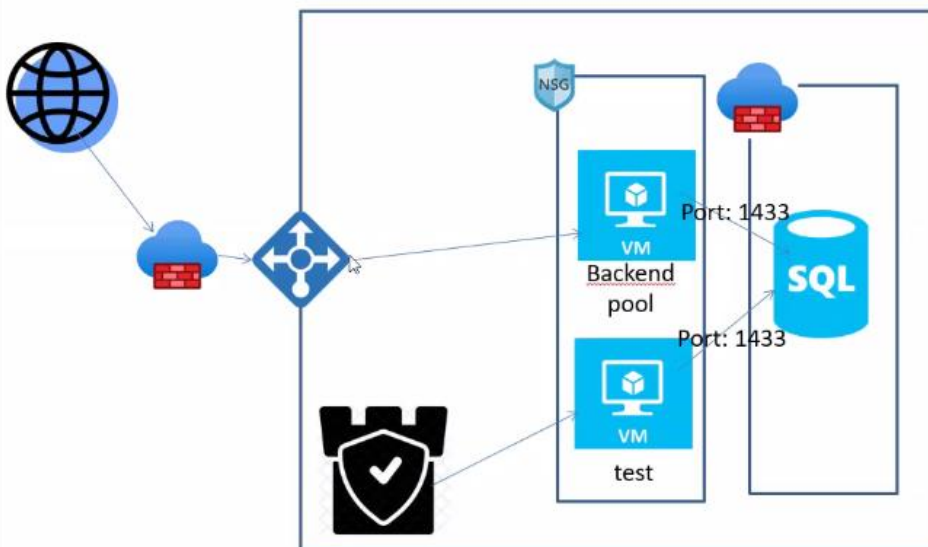
- 3) Utilizzare le **giuste interfacce**:

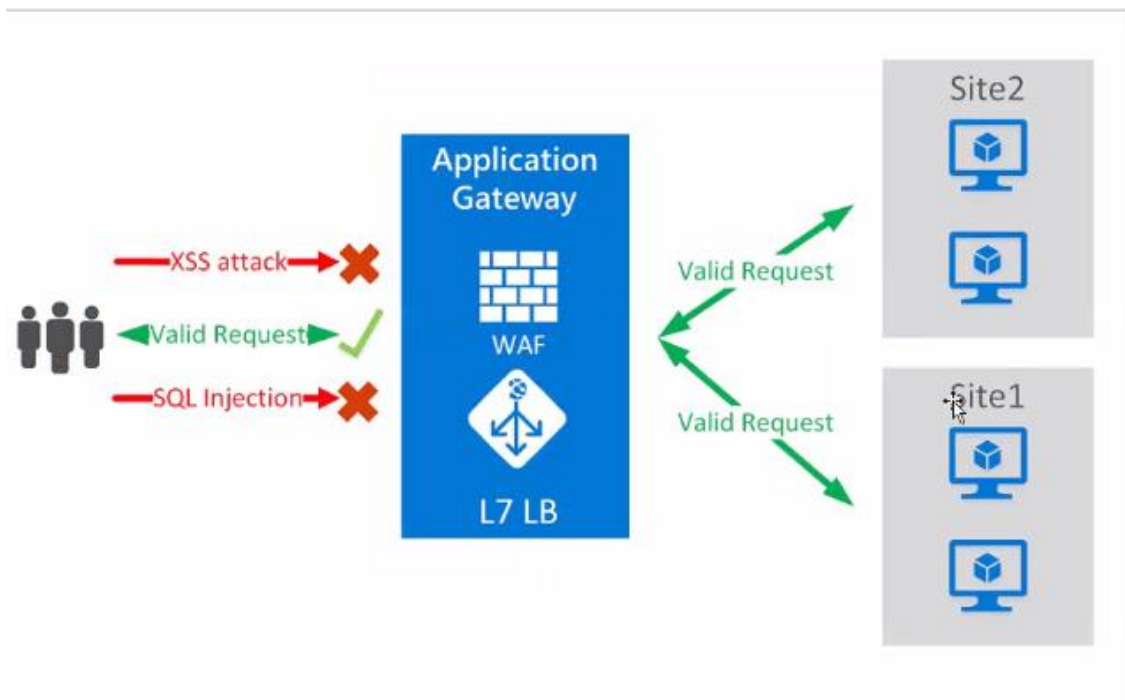
Gli oggetti sono “conosciuti” solo per il proprio insieme di operazioni.

DP definiscono cosa mettere/non nell’interfaccia

CLOUD: Sito Web

Un servizio di bilanciamento del carico interno viene distribuito con un indirizzo IP front-end privato. I pacchetti in ingresso arrivano all'indirizzo IP pubblico del firewall, vengono convertiti nell'indirizzo IP privato del servizio di bilanciamento del carico e quindi tornano all'indirizzo IP privato del firewall usando lo stesso percorso di ritorno.





Si occupano di questo i Cloud Architect, noi usiamo il cloud per caricarci i nostri software (li carichiamo su macchine virtuali).

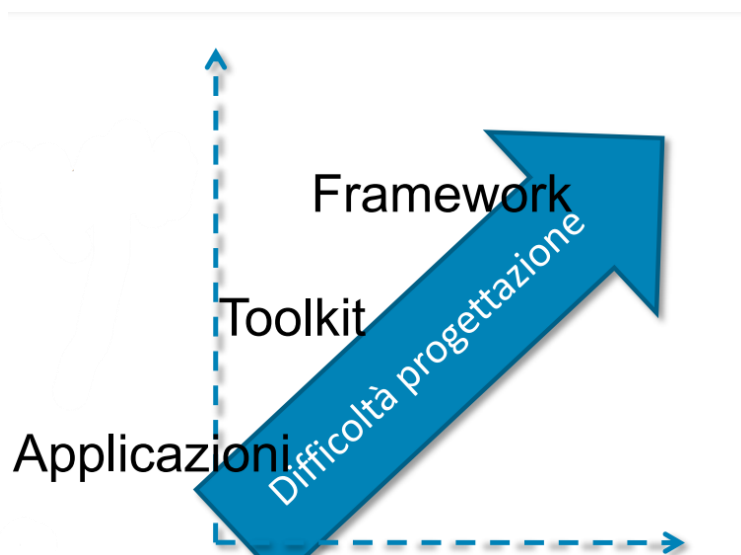
4) Livelli di Progettazione

Progettare applicazioni:

- ➔ Riutilizzo "interno"
- ➔ Riduzione delle dipendenze

Toolkit (librerie software):

- ➔ Insieme di oggetti correlati e riutilizzabili progettati per fornire funzionalità generiche.
- ➔ Riutilizzo del codice (code reuse): C++/Java I/O stream library



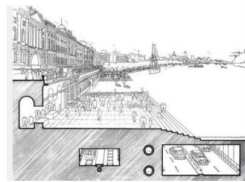
Framework:

- Insieme di classi che cooperano per costruire architetture riutilizzabili per sviluppare un dominio di applicazioni.
- Impone un **disegno architettuale**.
 - Riutilizzo architettuale.
- “Inversion of control”
- Minor grado di accoppiamento possibile.
- NON sono design pattern! Infatti:
 - DP sono più astratti
 - DP disegnano architetture più piccole
 - DP sono meno specializzati

Tipologie di Pattern:

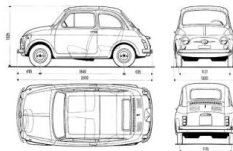
■ Architetture

- Stili architetture di alto livello



■ Progettuali

- Definiscono micro architetture



■ Idiomi

- Risolvono piccoli problemi
 - Legati al linguaggio



Design Pattern Architetture:

- Pattern di alto livello.
 - Guida nella scomposizione in sottosistemi.
 - Ruoli e responsabilità
 - Regole di comunicazione fra le componenti
- Agglomerati di pattern architetture
- **Esempi:**
 - Paradigma client-server
 - Model-View-Controller (MVC)
 - Peer to peer

Design Pattern Progettuali:

Progettazione di dettaglio di componenti:

Definiscono micro-architetture: schema di comunicazione fra gli elementi di un sistema software.

Basi per costruire i pattern architetturali:

Sono i pattern (Gang of Four): GoF

- Factory
- Command
- Proxy
- Observer

Design Pattern e Idiomi:

- Basso livello di astrazione.
- Specifici del linguaggio di programmazione.
- Utilizzano direttamente direttive del linguaggio.
- Design Pattern:
 - Soluzione generica ad una classe di problemi
 - Propone un modello architetturale.
- Idioma:
 - Soluzione specifica ad un linguaggio
 - Legato alla tecnologia

Un design idiomatico rinuncia alla genericità della soluzione basata su un pattern a favore di una implementazione che poggia sulle caratteristiche (e le potenzialità) del linguaggio.

Utilizzare i Design Pattern:

Come selezionare i DP?

- Considerare come il DP risolve il problema.
- Conoscere le relazioni fra DP
- Considerare le possibili cause di cambiamento dell'architettura
- Considerare cosa può evolvere nell'architettura.

Come utilizzare un design pattern?

- 1) Leggere il DP attentamente
- 2) Rileggere le sezioni che descrivono Struttura, Partecipanti e Collaborazioni.
- 3) Analizzare il codice di esempio fornito.
- 4) Scegliere nomi appropriati per i partecipanti.
- 5) Definire la struttura delle classi del pattern all'interno della propria applicazione.
- 6) Scegliere nomi appropriati per le operazioni.
- 7) Implementare le operazioni

Anti-Pattern

Problemi ricorrenti che si incontrano durante lo sviluppo dei programmi e che dovrebbero essere evitati, non affrontati!

Evitare gli errori già commessi

Evidenziare perché una soluzione sbagliata può sembrare in principio corretta per un problema.

Descrivere come passare dalla soluzione sbagliata a quella corretta.