INTRODUZIONE AI DESIGN PATTERN
INGEGNERIA DEL SOFTWARE

Università degli Studi di Padova
Dipartimento di Matematica
Corso di Laurea in Informatica, A.A. 2013 – 2014

rcardin@math.unipd.it

SOMMARIO

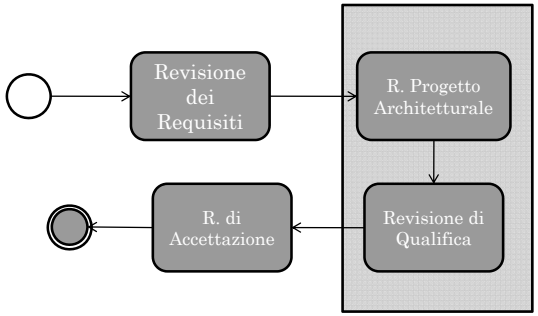
- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

SOMMARIO

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

DESIGN PATTERN

- Specifica Tecnica, Definizione di Prodotto, Piano di Qualifica



INTRODUZIONE

- Progettare software *object oriented* è difficile
  - Riusabilità
    - Ci vuole molta esperienza
  - Soluzioni comuni e ricorrenti (*pattern)*
    - Flessibili, eleganti, riusabili

- Non solo progettazione di *software*
  - Macbeth, novelle romantiche
- Altruismo …
  - Si tende a tenere per se le soluzioni vincenti!

INTRODUZIONE

- Design Pattern
  - Raccolta dell'esperienza dei progettisti, presentata in un modo effettivamente utilizzabile
    - Riutilizzo di soluzioni architetturali vincenti
  - Nessuna soluzione nuova (o non documentata)
  - Non vogliamo riscoprire la ruota?!

SOMMARIO

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

COS'È UN DESIGN PATTERN

Definizione

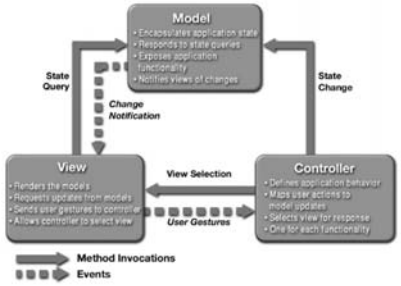
"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

Christopher Alexander

- 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

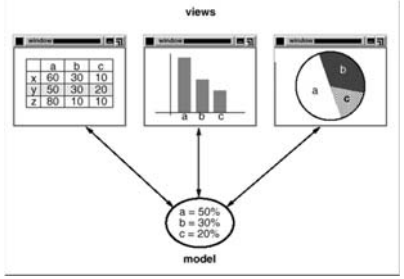
Design Pattern: Esempio

- 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



Design Pattern: Esempio

- MVC: Model – View
  - Disaccoppiamento → protocollo di notifica
    - *View* deve garantire una visualizzazione consistente
  - Diversi *design pattern*
    - Observer design pattern
    - Composite design pattern



Design Pattern: Esempio

- MVC: View – Controller
  - Permette la modifica del comportamento in risposta all'*input*
    - Nessuna modifica di visualizzazione
  - Strategy/Algorithm design pattern
- Altri pattern…
  - Factory Method / Template design pattern
  - Decorator 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 dei 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 Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

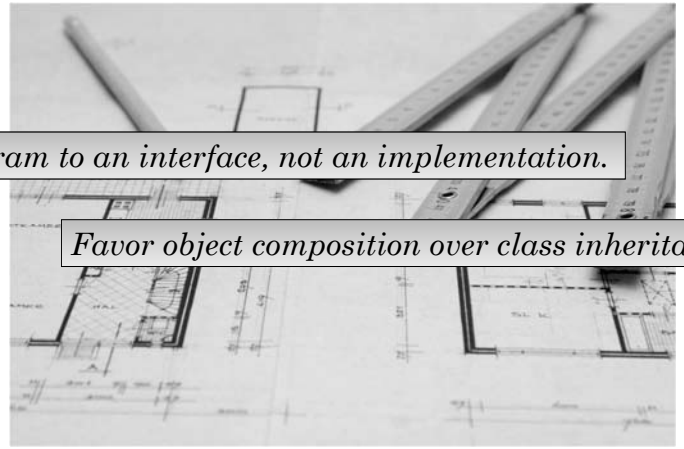
Sommario

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

Risolvere Problemi Architetturali

- Utilizzare gli oggetti appropriati
  - È difficile decomporre il sistema in oggetti!
    - Oggetti "reali" ≠ oggetti "non reali"
  - DP permettono di identificare le astrazioni meno ovvie
- Utilizzare la giusta granularità
  - Come decidere cosa deve essere un oggetto?
  - DP forniscono la granularità più appropriata
- Utilizzare le giuste interfacce
  - Gli oggetti sono "conosciuti" solo per il proprio insieme di operazioni
  - DP definiscono cosa mettere/non nell'interfaccia

Principi di Progettazione



Program to an interface, not an implementation.

Favor object composition over class inheritance

Principi di Progettazione

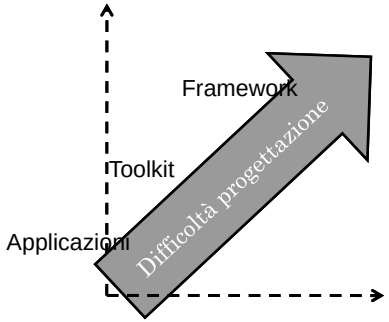
- Progettare pensando al cambiamento
  - Anticipazione di nuovi requisiti e loro estensioni
    - Massimizza il riuso
  - DP permettono al sistema di evolvere
    - Creazione di oggetti indirettamente (interfacce)
    - Decidere quale operazione utilizzare solo a *runtime*
    - Limitare le dipendenze dall'ambiente di esecuzione
    - Nascondere l'implementazione degli oggetti
    - Isolare gli algoritmi che hanno possibilità di essere estesi
    - Utilizzare astrazione e livellamento delle classi per ottenere un sistema con un basso grado di accoppiamento
    - Aggiungere nuove funzionalità utilizzando la composizione

Sommario

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

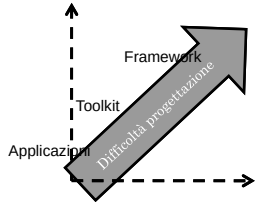
Livelli di Progettazione

- Progettare applicazioni
  - Riuso "interno"
    - Riduzione delle dipendenze



- Toolkit (librerie *software*)
  - Insieme di oggetti correlati e riusabili progettati per fornire funzionalità generiche.
  - Riutilizzo del codice (*code reuse*)
    - C++/Java I/O stream *library*

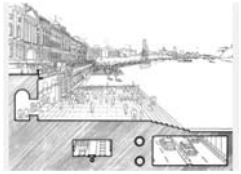
Livelli di Progettazione



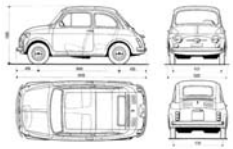
- Framework
  - Insieme di classi che cooperano per costruire architetture riutilizzabili per sviluppare un dominio di applicazioni
  - Impone un disegno architetturale
    - Riutilizzo architetturale
  - *"Inversion of control"*
  - Minor grado di accoppiamento possibile
  - NON sono *design pattern*
    - DP sono più astratti
    - DP disegnano architetture più piccole
    - DP sono meno specializzati

Tipologie di Pattern

- Architetturali
  - Stili architetturali di alto livello



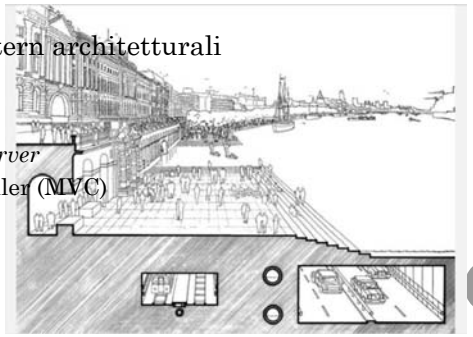
- Progettuali
  - Definiscono micro architetture



- Idiomi
  - Risolvono piccoli problemi
    - Legati al linguaggio

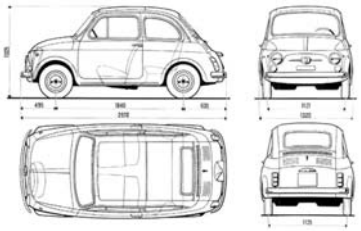
Design Pattern Architetturali

- Pattern di alto livello
  - Guida nella scomposizione in sottosistemi
    - Ruoli e responsabilità
    - Regole di comunicazione fra le componenti
  - Agglomerati di pattern architetturali
  - 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: Idiomi

- Basso livello di astrazione
  - Specifici del linguaggio di programmazione
    - Utilizzano direttamente direttive del linguaggio
  - Esempio
    - Come implementare un Singleton in Java?
      - Definire un attributo statico e privato
      - Implementare il metodo pubblico e statico getInstance
      - Impostare il costruttore privato.

Design Pattern e Idiomi

- 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.

Sommario

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

Utilizzare i Design Pattern

- Come selezionare i DP?
  - Considerare come il DP risolve il problema
  - Leggere la sezione delle motivazioni
  - Conoscere le relazioni fra DP
  - Considerare le possibili cause di cambiamento dell'architettura
  - Considerare cosa può evolvere nell'architettura

Utilizzare i Design Pattern

- Come utilizzare un *design pattern*
  - Leggere il DP attentamente
  - Rileggere le sezioni che descrivono Struttura, Partecipanti e Collaborazioni
  - Analizzare il codice di esempio fornito
  - Scegliere nomi appropriati per i partecipanti
  - Definire la struttura delle classi del *pattern* all'interno della propria applicazione
  - Scegliere nomi appropriati per le operazioni
  - 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

- Esempi
  - Reinventare la ruota (quadrata)
  - Codice spaghetti
  - Fede cieca
  - Anomalia della sottoclasse vuota
  - Programmazione "copia e incolla"
  - …

RIFERIMENTI

- Design Patterns, Elements of Reusable Object Oriented Software, GoF, 1995, Addison-Wesley
- MVC
  http://www.claudiodesio.com/ooa&d/mvc.htm
- Design Patterns
  http://sourcemaking.com/design_patterns