



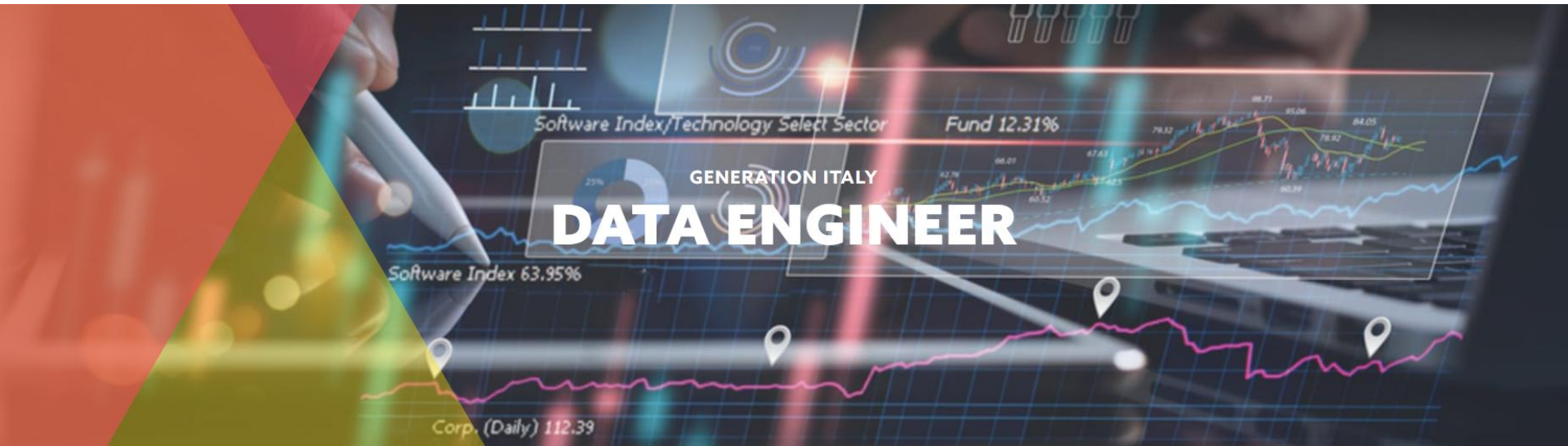
ACCADEMIA
DEL LEVANTE
WWW.ACCADEMIADELLELEVANTE.ORG

Generation
ITALY



IASEM
Istituto Ali Studi Euro Mediterranei

Data Engineer



Introduzione al design pattern

Franchini Roberto

Sommario

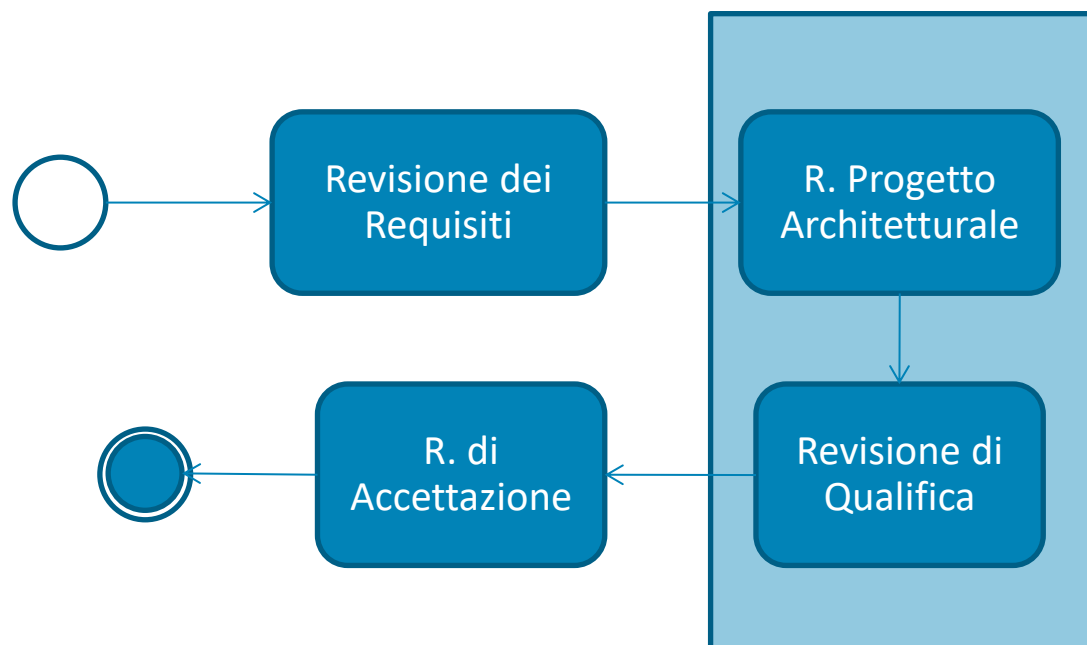
- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

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

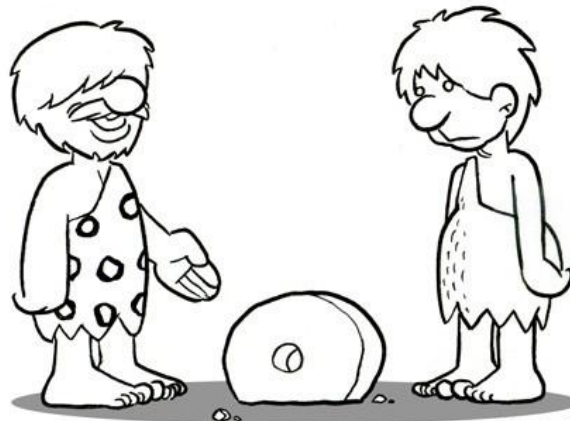
- Altruismo ...
 - Si tende a tenere per se le soluzioni vincenti!



Introduzione

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

- HO INVENTATO LA RUOTA SGONFIA.



MASSY

Ad uso esclusivo degli allievi del corso Data Engineer di Generation Italy. Vietata la diffusione all'esterno.

Sommario

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

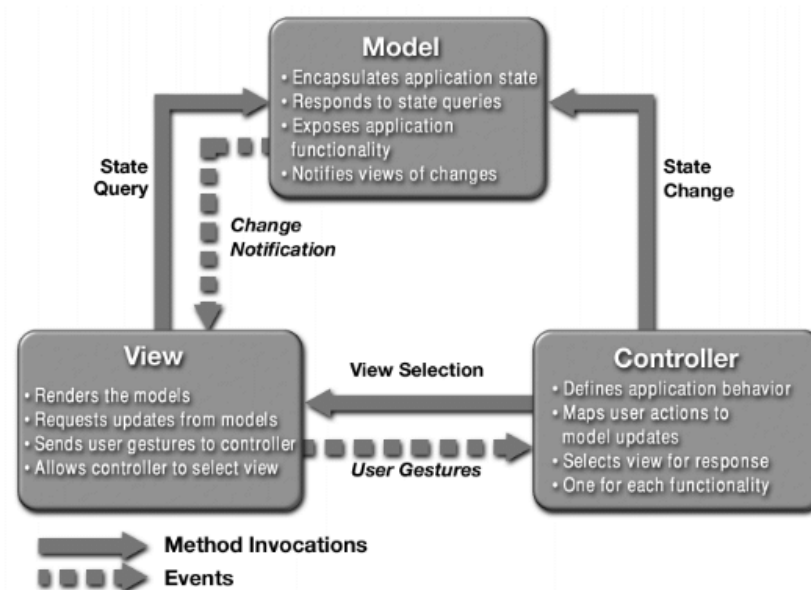
Cos'è un Design Pattern

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

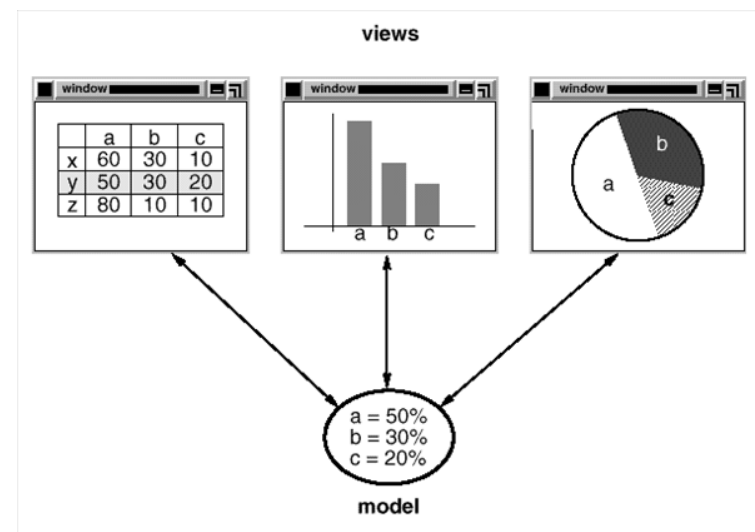
Design Pattern: Esempio

- 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 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

Sommario

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

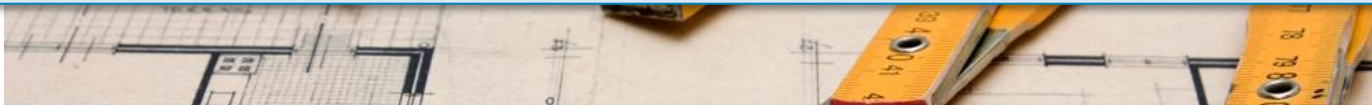
Risolvere Problemi Architeturali

- 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

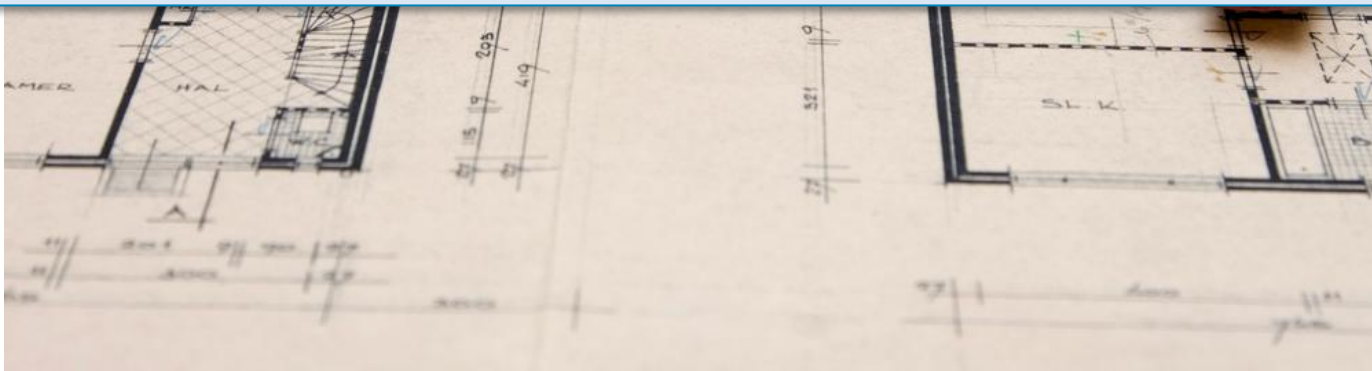
Principi di Progettazione



Programma su un'interfaccia, non su un'implementazione.



Privilegiare la composizione degli oggetti rispetto all'ereditarietà delle



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**

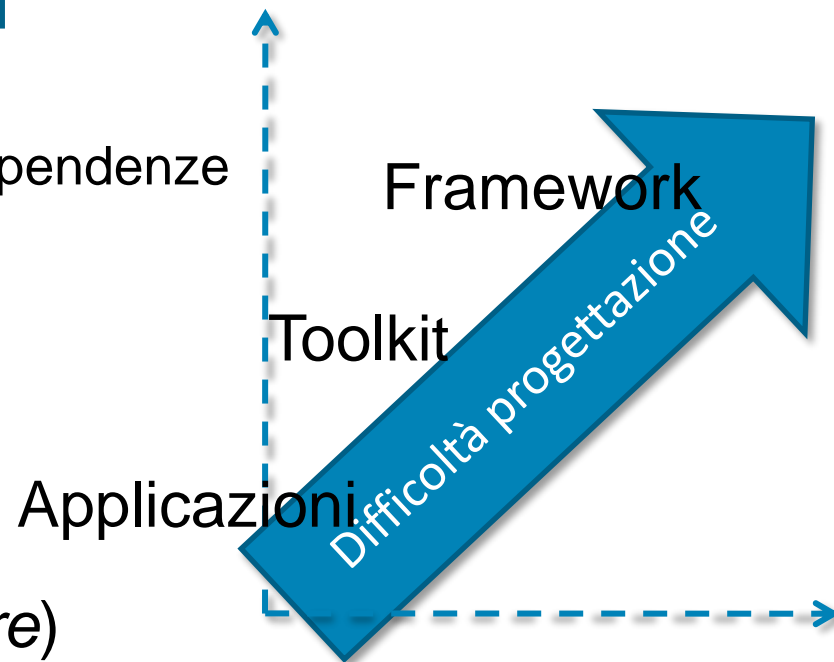
Sommario

- 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



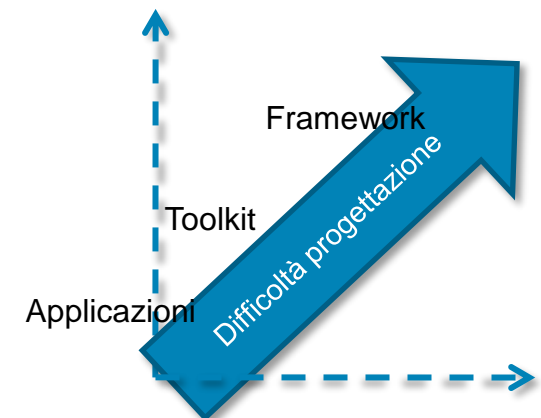
- Toolkit (librerie software)

- 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

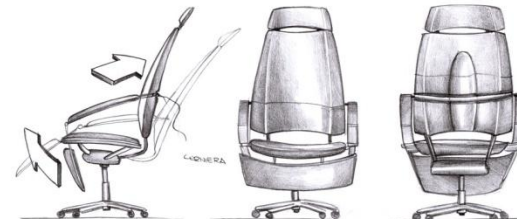
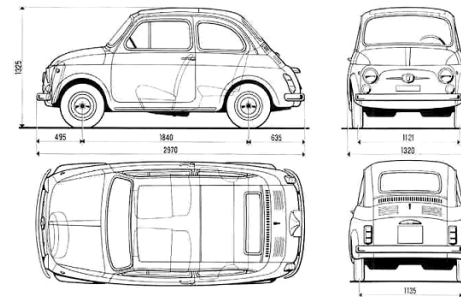
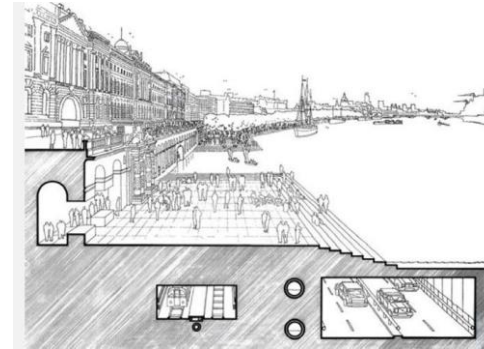
■ Framework

- 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



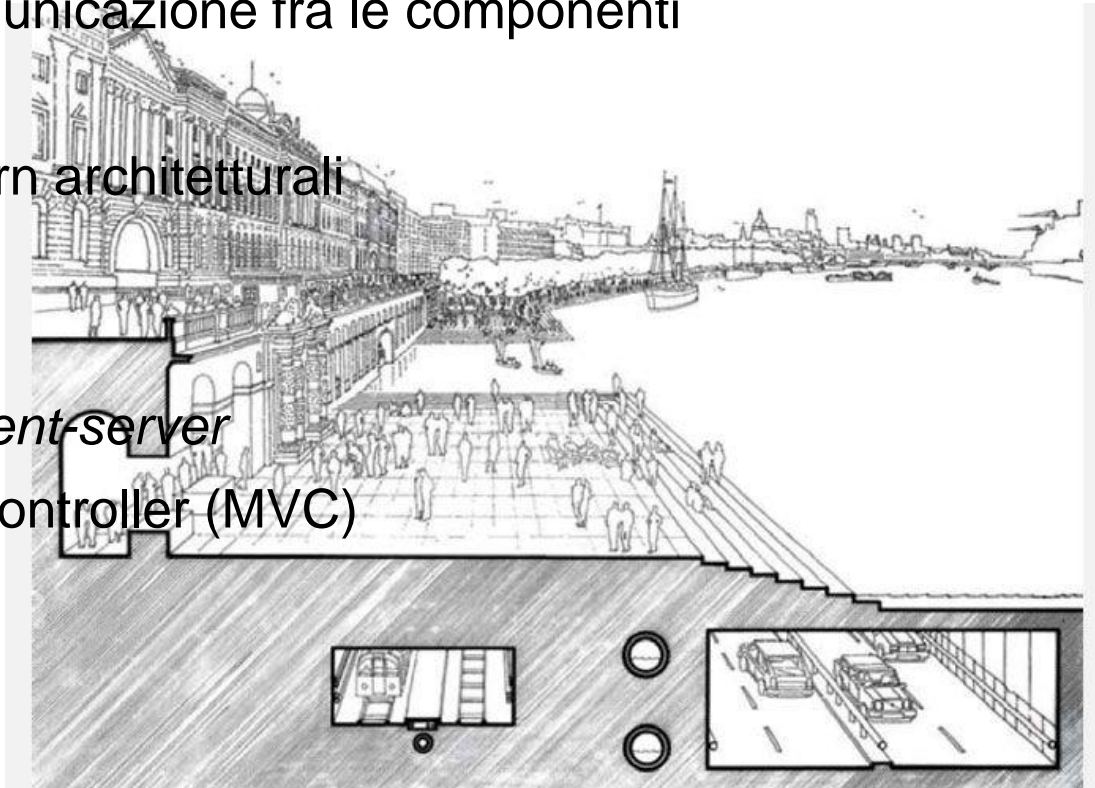
Tipologie di Pattern

- Architetture
 - Stili architetture di alto livello
- Progettuali
 - Definiscono micro architetture
- Idiomi
 - Risolvono piccoli problemi
 - Legati al linguaggio



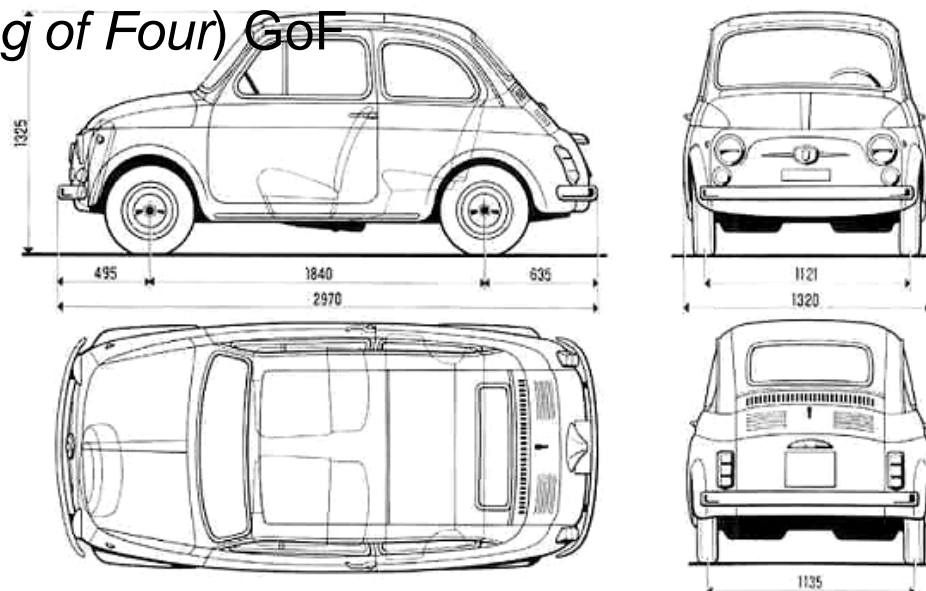
Design Pattern Architeturali

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



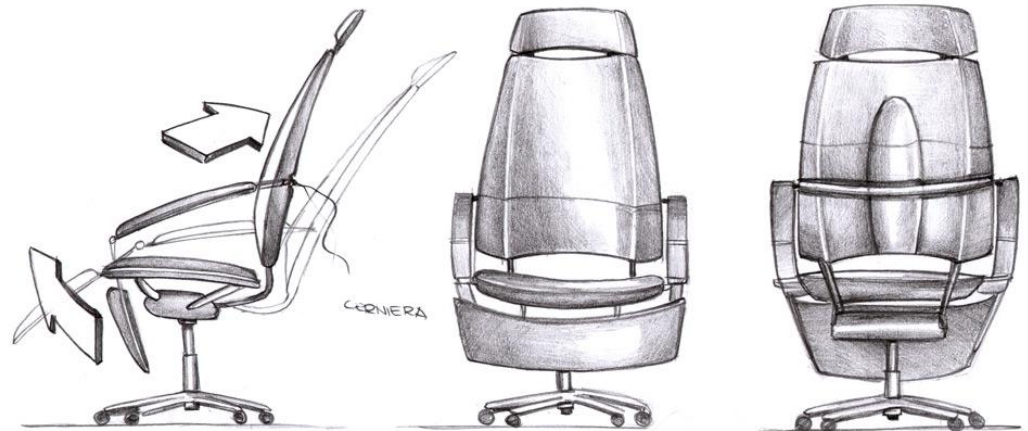
Design Pattern Progettuali

- 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 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**
 - 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

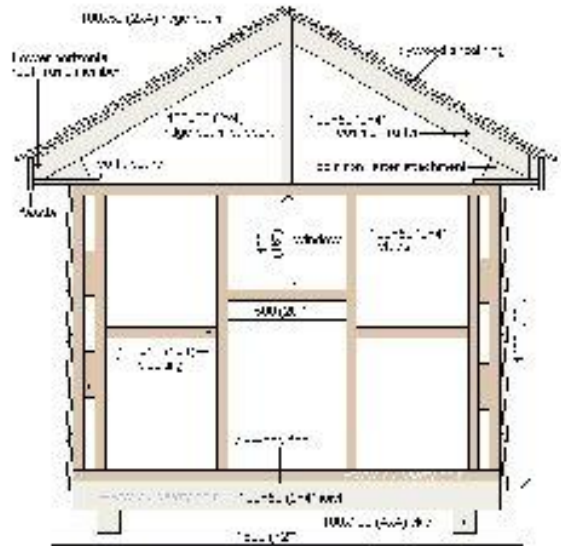
- Design Patterns, Elements of Reusable Object Oriented Software, GoF, 1995, Addison-Wesley
- MVC <http://www.claudiodesio.com/ooa&d/mvc.htm>
- Design Patterns
http://sourcemaking.com/design_patterns

Unified Modeling Language

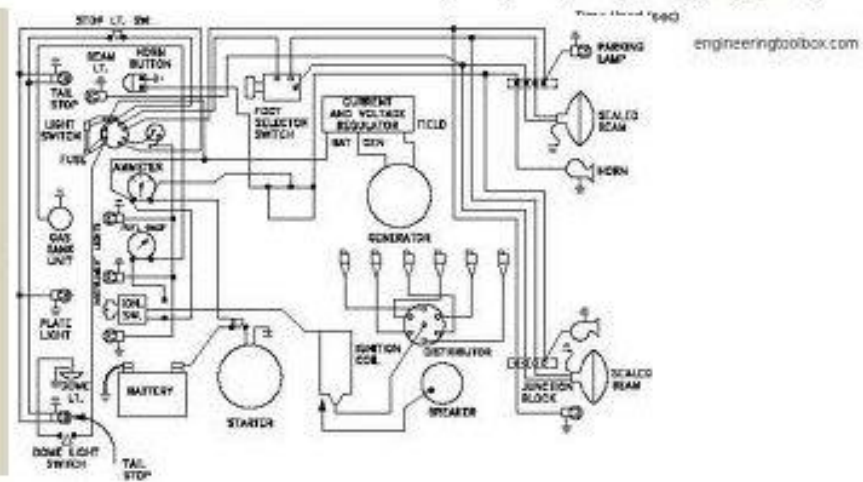
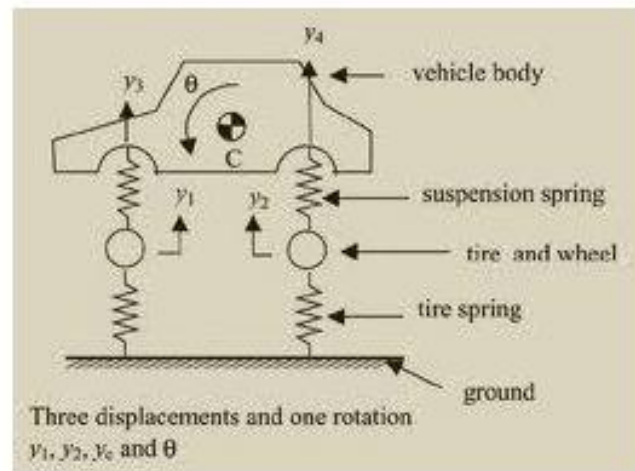
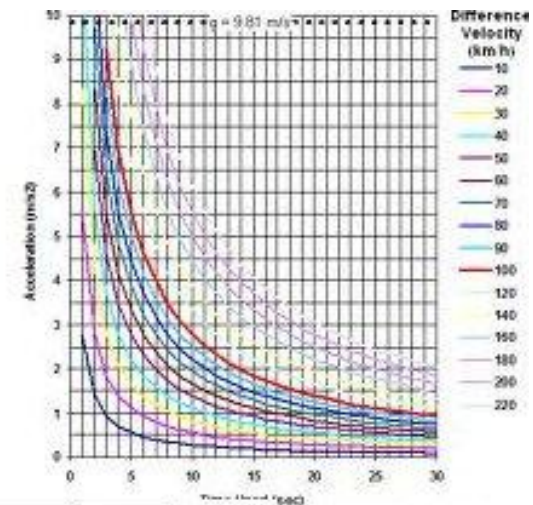
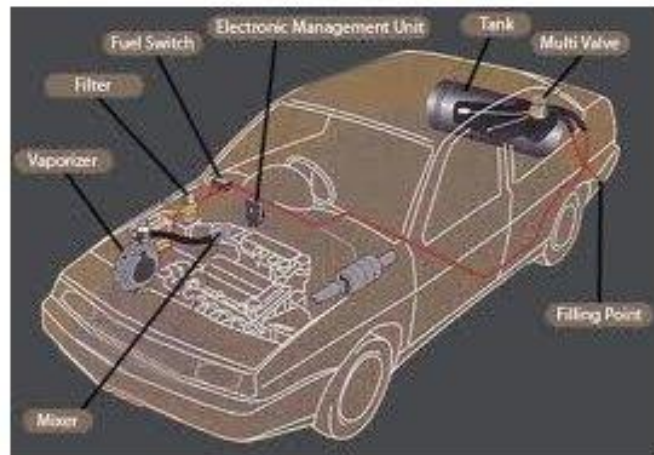
UML

Modeling

La maggior parte dei processi di produzione e costruzione richiedono la creazione di un modello prima della realizzazione del prodotto vero e proprio.



Automobile models



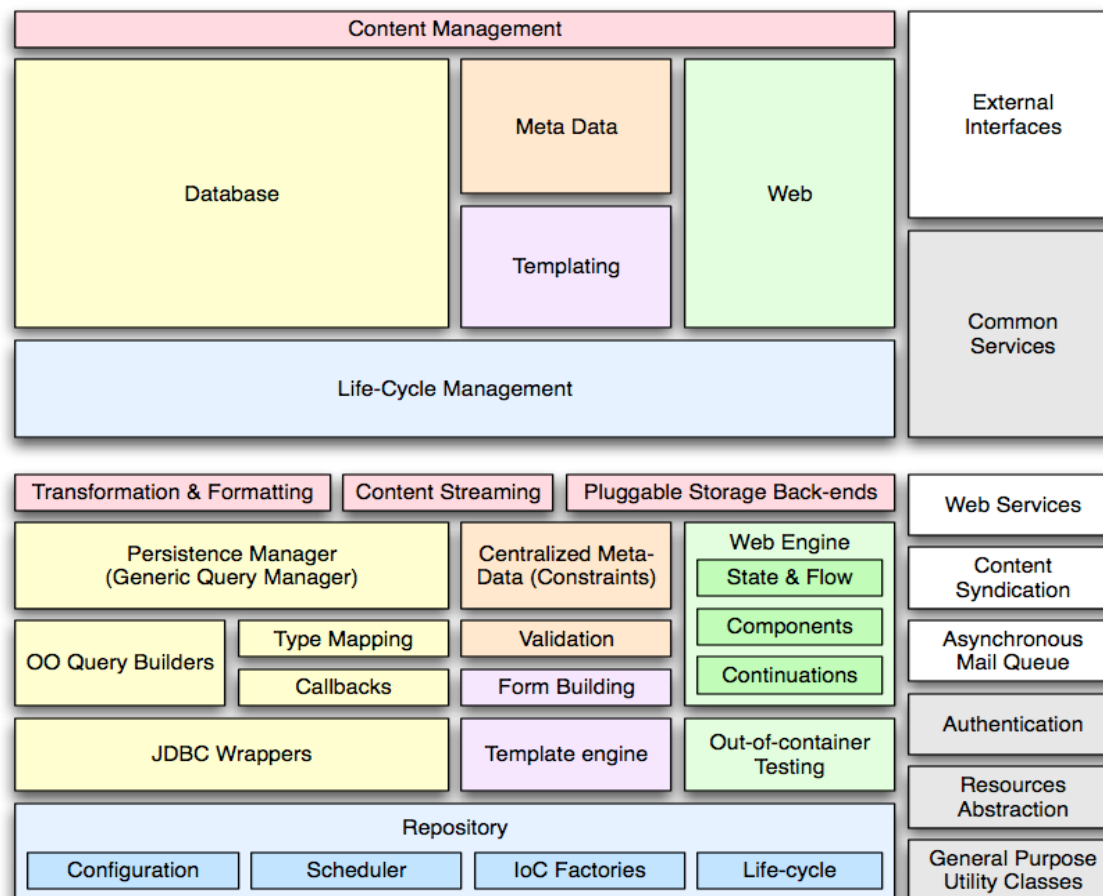
Perchè il modello è importante?

Per una Migliore comunicazione con tutti gli attori del Progetto coinvolti



Perchè il modello è importante?

Per documentare decisioni di progettazione di alto livello



Perchè il modello è importante?

Per ridurre i rischi



Perchè il modello è importante?

Per l'analisi what-if



Perchè il modello è importante?

Consente il riutilizzo di componenti e modelli di progettazione tra progetti



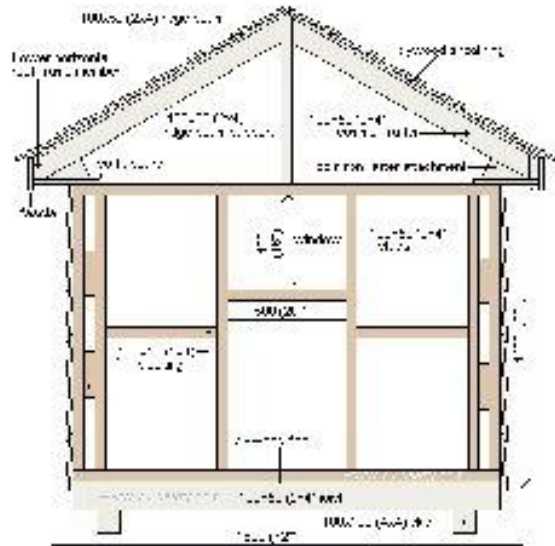
Perchè il modello è importante?

Se non fai la modellazione, potresti...

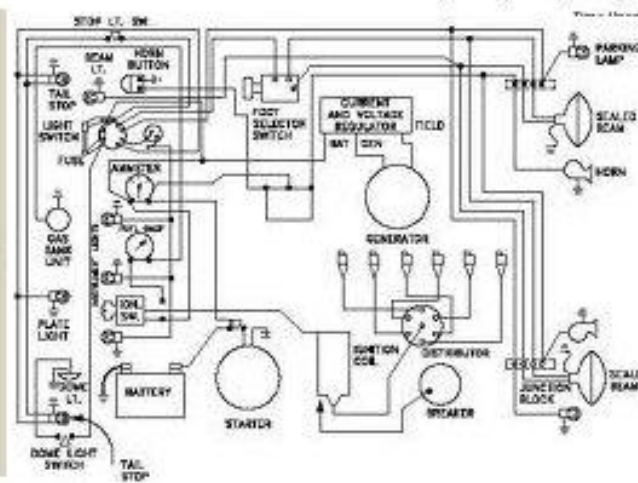
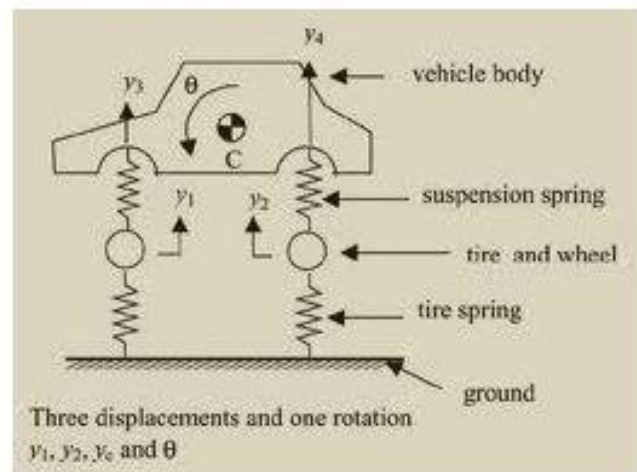
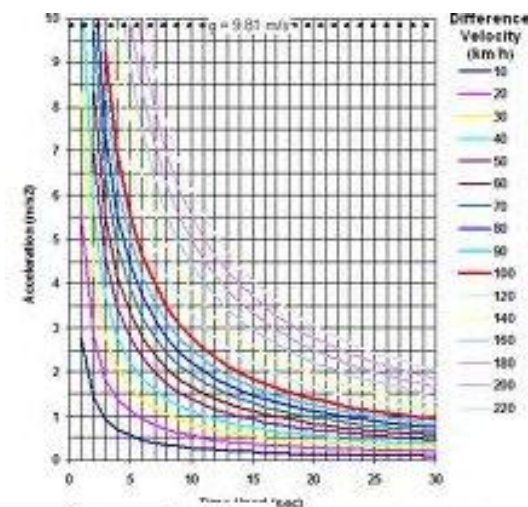
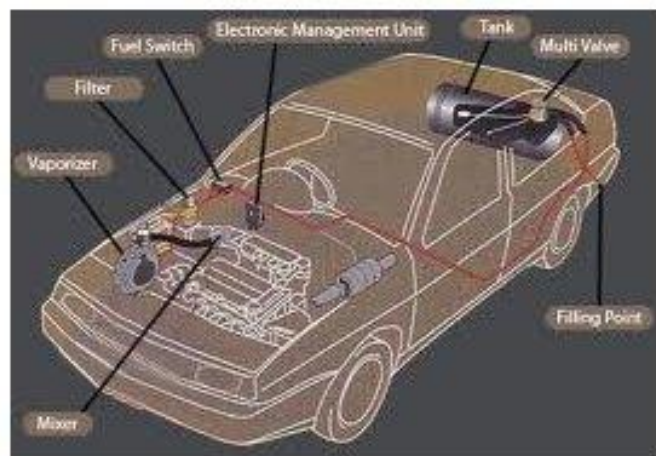


Differenti viste di un sistema

Ogni sistema può essere visto da più prospettive o punti di vista



Differenti viste di un sistema



Viste in un Software

- Modules
- Classes and their relations
- Real time interactions
- Data flow
- Deployment
- User Interface
- ... etc



UML - Unified Modeling Language

Obiettivi

Presentare un approccio visuale alla progettazione

Illustrare i vantaggi dell'utilizzo di diagrammi nella fase di progettazione

Rispondere alla domanda: cos'è UML?

Descrivere il Processo Unificato di sviluppo del software (Unified Process)

Descrivere la struttura di UML

Presentare l'utilizzo dei casi d'uso per modellare il dialogo tra utilizzatore e sistema

Descrivere i principali diagrammi di UML

Presentare un caso di studio con l'applicazione di UML



Un approccio visuale alla progettazione

- Per quale motivo è utile un approccio visuale alla progettazione?
- Chi progetta un qualsiasi tipo di costruzione o artefatto utilizza sempre figure, schemi, diagrammi per svolgere la propria attività:
 - ingegneri, architetti, ma anche stilisti utilizzano diagrammi e figure per visualizzare i propri progetti
- Anche i progettisti e gli analisti di sistemi informativi utilizzano figure e diagrammi per visualizzare il risultato del loro lavoro:
 - un sistema software
 - un sistema informativo
- *Ciò avviene anche se il tipo di prodotto finale che risulta dalla progettazione non è necessariamente visuale*

Vantaggi dell'utilizzo di diagrammi nella fase di progettazione

- Sia che si progetti un edificio sia che si progetti un sistema software il progettista ha la necessità di rappresentare i diversi aspetti del progetto
 - si utilizzano diagrammi differenti, ognuno focalizzato su uno o più aspetti
 - Nel caso dell'edificio
 - alcuni disegni rappresentano una visione completa da diversi punti di vista
 - altri alcuni particolari come ad esempio gli impianti tecnologici
 - Allo stesso modo per un sistema informativo
 - un diagramma può rappresentare i collegamenti tra le componenti
 - altri alcuni particolari come ad esempio la sequenza delle comunicazioni tra le componenti
 - Si tratta di applicare il concetto di “astrazione” attraverso il quale una realtà anche molto complessa viene rappresentata semplificandola in un *modello*

Generazione dei modelli

- Durante le fasi di analisi e progettazione vengono generati dei modelli che consentono di identificare e separare le caratteristiche (utili al progetto) di un sistema reale (ad esempio una classe che modella l'oggetto cliente)
- Il progettista dovrà quindi decidere quali caratteristiche sono rilevanti per il sistema che stà costruendo, inserirle nel modello e definire le relazioni tra gli elementi del modello

Tipi di relazione

- Vi sono diversi tipi di relazione da considerare:
 - Strutturali, tra elementi interdipendenti
(ad esempio associazioni tra classi)
 - Temporali, per rappresentare sequenze di eventi nel tempo
(ad esempio messaggi sequenziali in un diagramma di interazione)
 - Causa-effetto, per definire precondizioni ad una determinata funzione
(ad esempio stati di un diagramma di stato)
 - Organizzative, per raggruppare opportunamente elementi del sistema
(ad esempio package di elementi)
 - Evolutive, per rappresentare le derivazioni tra elementi del modello
nel tempo
(ad esempio dipendenze tra diagrammi del modello)

Cos'è UML

- Unified Modeling Language (UML) è un linguaggio di modellazione visuale
- *E' uno strumento per analisti e progettisti di sistemi orientati agli oggetti che consente di modellare, rappresentare e documentare sistemi software*
 - UML non è un linguaggio di programmazione, non è uno strumento di CASE
- Vi sono strumenti di CASE che possono generare codice in diversi linguaggi a partire da modelli UML
 - si tratta del codice di struttura di oggetti che poi richiedono da parte dello sviluppatore la scrittura manuale del codice che implementa i metodi
 - UML non è una metodologia di sviluppo del software
- Molte metodologie di analisi e progettazione, pur mantenendo diversi procedimenti, hanno standardizzato la loro notazione per rappresentare visivamente i modelli di un sistema con UML

Cos'è UML

- UML è un linguaggio, un insieme di elementi e di regole, di specifica formale
- Gli elementi sono forme grafiche (linee, rettangoli, ecc.) che rappresentano ciò che si stà modellando
- Le regole che spiegano come combinare gli elementi sono di tre tipi:
 - sintassi astratta,
espressa tramite diagrammi e linguaggio naturale
 - regole sintattiche,
esprese tramite Object Constraint Language (OCL) e linguaggio naturale
 - semantica,
espressa in linguaggio naturale con il supporto di diagrammi

Origini e breve storia di UML (1)

- ⑩ La Rational Software Corporation e l'Object Management Group (OMG)
 - hanno unificato gli elementi rappresentativi di tre diverse metodologie di rappresentazione di diagrammi orientati agli oggetti
- ⑩ Le tappe principali dell'evoluzione di UML
 - ✎ 1990-1994, affermazione di tre metodologie per lo sviluppo ad oggetti:
 - Rumbaugh (Object Modeling Technique - OMT), Booch, Jacobson (Object-Oriented Software Engineering – OOSE)
 - ✎ 1994, unificazione delle metodologie Rumbaugh e Booch
 - insieme alla Rational Software Corporation
 - ✎ 1995, le due metodologie danno origine allo Unified Method
 - ✎ 1995, unione della compagnia di Jacobson (Objectory) alla Rational, inizio dello sviluppo di UML
 - ✎ 1996, formazione del consorzio UML Partners
 - (Rational Software Corporation, IBM, HP, Microsoft, Oracle)

Origini e breve storia di UML (2)

- Le tappe principali dell'evoluzione di UML (continua)
 - 1997, versione 1.1 di UML che viene aggiunto alle tecnologie adottate dall'Object Management Group (OMG)
 - 1998, versione 1.2 delle specifiche UML
 - 2001, versione 1.4
 - ...
- UML è il linguaggio standard di modellazione più diffuso nello sviluppo di software a livello industriale
 - E' uno standard in evoluzione riconosciuto dall'Organizzazione Internazionale per la Standardizzazione (International Organization for Standardization, ISO)
 - www.uml.org

The screenshot shows a web browser window with the address bar displaying `www.uml.org`. The page features the UML logo (Unified Modeling Language) and a navigation menu with links: HOME, TRAINING, VENDORS, OMG WEB, MEMBERSHIP, and CONTACTS. The main content area is titled "ADOPTED PROFILES UML RESOURCES" and includes a sub-header "This is a useful link to UML® resources." with a "READ MORE" button. Below this, there is a horizontal menu with links: WHAT IS UML?, UML VENDOR, UML RESOURCES (highlighted), UML SPECIFICATIONS, OMG UML CERTIFICATION, and TRAINING PAGE. The bottom section contains three white boxes with orange circular icons: "NEWS & ARTICLES" (document icon), "SUCCESS STORIES" (list icon), and "CURRENT SPECIFICATION" (cloud icon). Each box has a "READ MORE" button. The Windows taskbar at the bottom shows various application icons and the system clock indicating 14:34 on 31/10/2016.

Processo Unificato di sviluppo del software (Unified Process)

- UML è un linguaggio che ha lo scopo di definire in modo formale un sistema
- Gli sviluppatori di UML hanno messo a punto per gli sviluppatori di sistemi anche un modo di procedere nel processo di sviluppo utilizzando UML:
 - il Processo Unificato di sviluppo del software (Unified Software Development Process – USDP)
- *Il Processo Unificato coinvolge persone, progetti, strumenti, processi, prodotti: i partecipanti e gli sviluppatori coinvolti nel progetto di sviluppo di un sistema, seguono un determinato processo, utilizzando strumenti di ausilio nello sviluppo, generando prodotti software*
 - *E' un processo di sviluppo iterativo ed incrementale*

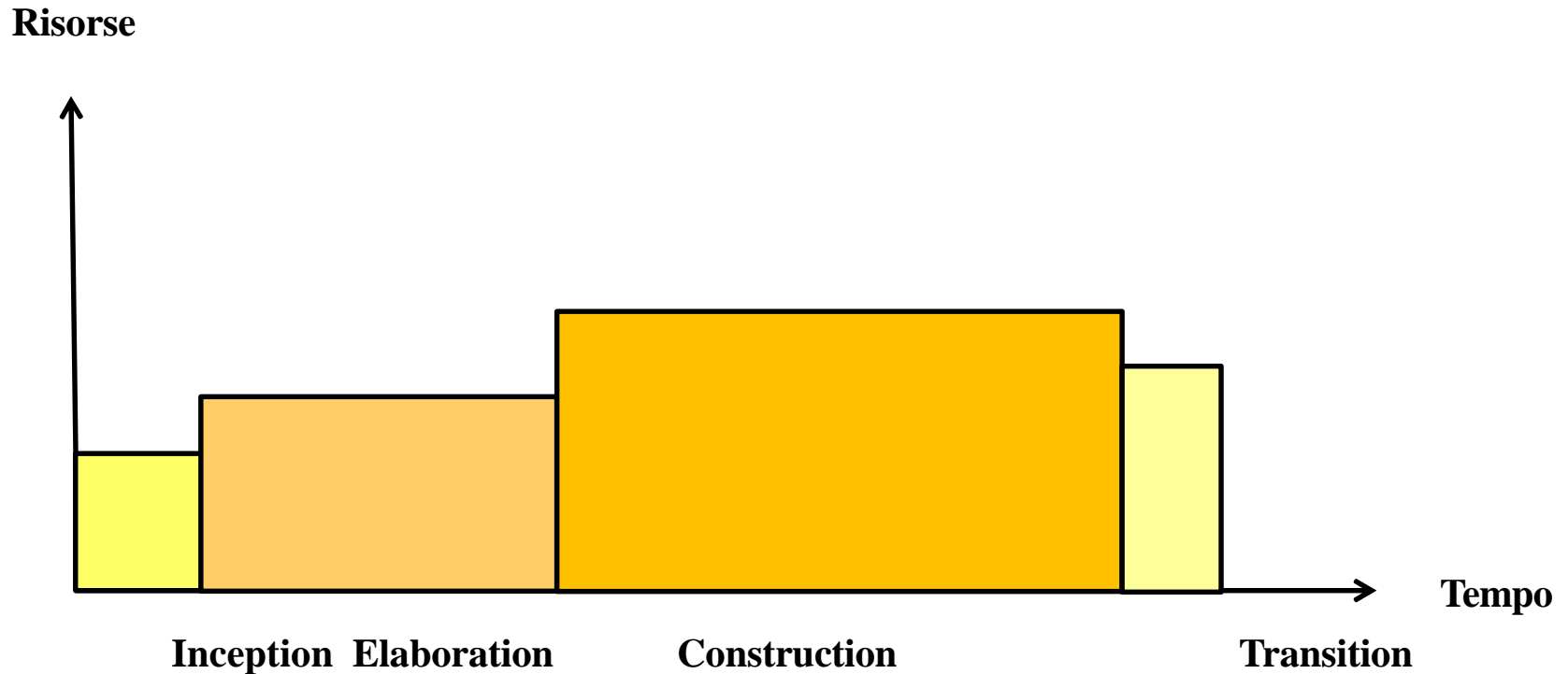
Caratteristiche del Processo Unificato

- Il processo parte dai requisiti dell'utente, che vengono raccolti nei cosiddetti “casi d'uso”, delle sequenze di esecuzione del sistema in grado di fornire un valore all'utente
- Dai casi d'uso gli sviluppatori producono i modelli del sistema e le implementazioni che li realizzano
 - Le caratteristiche del processo:
 - architetto-centrico: l'architettura del sistema viene sviluppata in modo da soddisfare i requisiti dei casi d'uso più importanti (piattaforma e struttura, sottosistemi)
 - iterativo: il progetto viene scomposto in sottoprogetti che costruiscono parti del sistema finale
 - incrementale: il sistema viene costruito incrementalmente unendo le singole parti sviluppate nei sottoprogetti

Caratteristiche del Processo Unificato

- L'attività è guidata dalla definizione dei requisiti funzionali espressi attraverso i casi d'uso
- Il caso d'uso descrive la funzionalità ed il dialogo fra l'attore che la utilizza ed il sistema che la fornisce
- Il processo definisce l'architettura di base e procede per incrementi successivi integrando le funzionalità richieste
- L'analisi individua i rischi nella realizzazione delle funzionalità
- Fasi del ciclo di vita (un progetto può essere costituito da più cicli)
 - *inizio (inception)*
 - *elaborazione (elaboration)*
 - *costruzione (construction)*
 - *transizione (transition)*
- Ogni fase può essere costituita da più iterazioni

Caratteristiche del Processo Unificato



*Profilo di un tipico progetto
che rappresenta qualitativamente
le grandezze relative delle quattro fasi dello Unified Process*

Caratteristiche del Processo Unificato

- Ogni fase è a sua volta caratterizzata da diverse attività eseguite ad ogni iterazione
 - *Business Modeling*
 - *Requirements*
 - *Analysis & Design*
 - *Implementation*
 - *Test*
 - *Deployment*
- Nelle fasi di inception ed elaboration prevalgono le attività relative ai requisiti e all'analisi, nella fase di construction prevalgono design, implementazione e test, nella transition prevalgono deployment e test
 - Cambia anche il contenuto delle iterazioni nelle fasi:
 - all'inizio ogni iterazione porta ad un perfezionamento dei casi d'uso e dell'architettura
 - poi le iterazioni sono finalizzate agli incrementi delle funzionalità

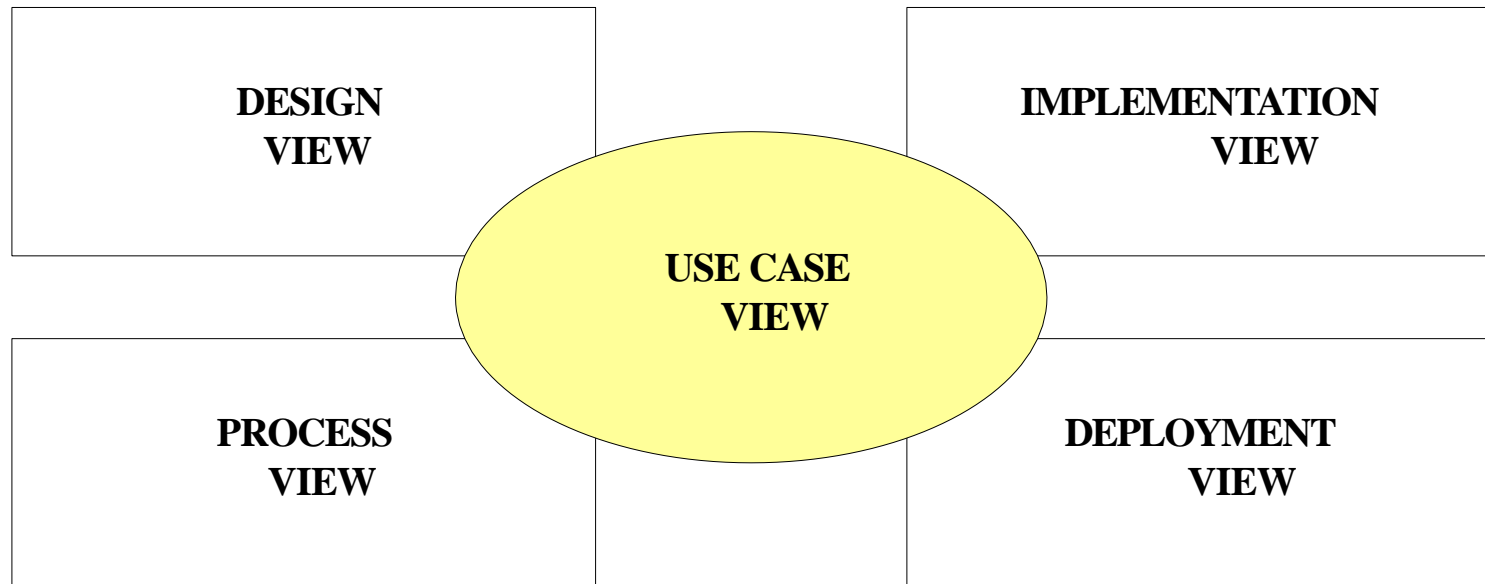
Prodotti del Processo Unificato

- I prodotti intermedi del processo sono i modelli (astrazioni delle caratteristiche del sistema)
- Ogni modello definisce il sistema da un certo punto di vista, i principali:
 - dei Casi d’Uso
 - di Analisi
 - di Progetto
 - di Implementazione
 - di Deployment
 - di Test

Struttura di UML

- In un approccio top-down all'UML si possono distinguere gli elementi fondamentali della sua struttura:
 - Viste, Diagrammi, Elementi del modello
- Le viste mostrano i differenti aspetti di un sistema attraverso la realizzazione di un certo numero di diagrammi
 - si tratta di astrazioni, ognuna delle quali analizza il sistema da modellare con un'ottica diversa (funzionale, non funzionale, organizzativa, ecc.), la somma di queste viste fornisce il quadro d'insieme
- I diagrammi permettono di esprimere le viste logiche per mezzo di grafici
 - vi sono diversi tipi di diagrammi destinati ad essere utilizzati ognuno per una particolare vista
- Gli elementi del modello sono i concetti che permettono di realizzare i vari diagrammi
 - indicano gli attori, le classi, i packages, gli oggetti, ecc.

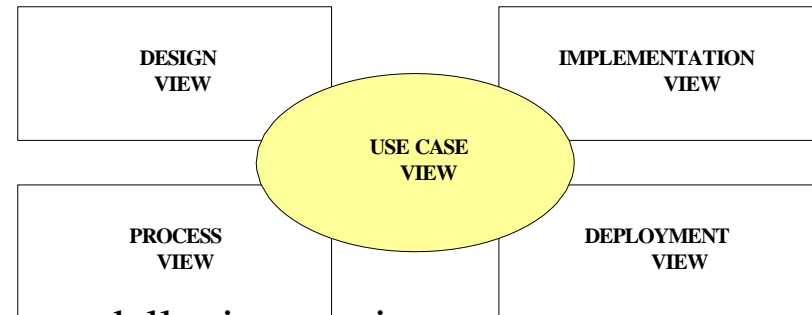
Le viste



Use Case View

- La “use case view” serve per analizzare i requisiti utente:
cosa il sistema dovrà fare

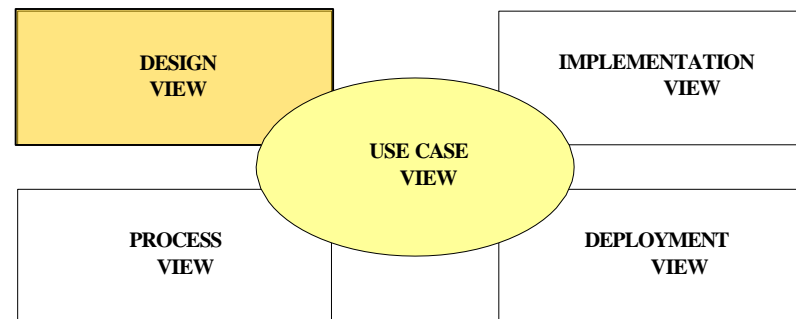
- Si tratta di una vista
ad alto livello
di importanza fondamentale



- guida lo sviluppo delle rimanenti
- stabilisce le funzionalità che il sistema dovrà realizzare
- Le funzionalità potranno essere individuate assieme al cliente grazie proprio all’ausilio di tale vista che coadiuva il reperimento dei requisiti
 - A questo livello di analisi,
è opportuno studiare il sistema considerandolo come una scatola nera,
concentrarsi sul *cosa fare* astraendosi il più possibile dal *come verrà fatto*

Design View

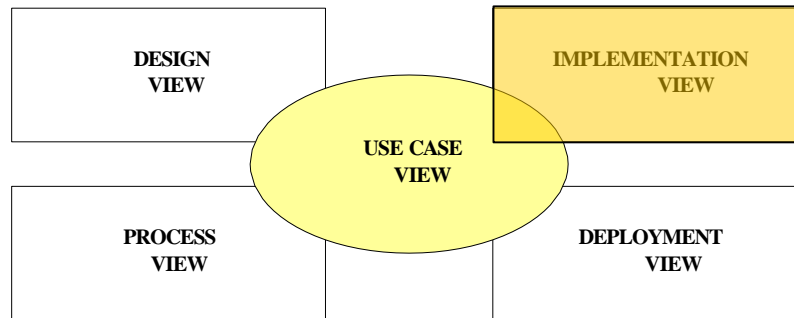
- La “desing view” descrive *come* le funzioni debbono essere realizzate



- In questa vista si analizza il sistema dall'interno
 - si trova la struttura statica del sistema (diagramma delle classi e diagramma degli oggetti)
 - si trova la collaborazione dinamica dovuta alle interazioni tra gli oggetti del sistema

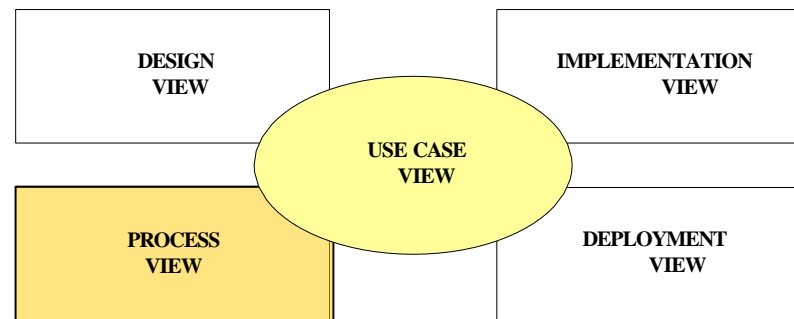
Implementation View

- La “implementation view” descrive come il codice contenuto nelle classi viene aggregato in moduli (package) e le relative interdipendenze



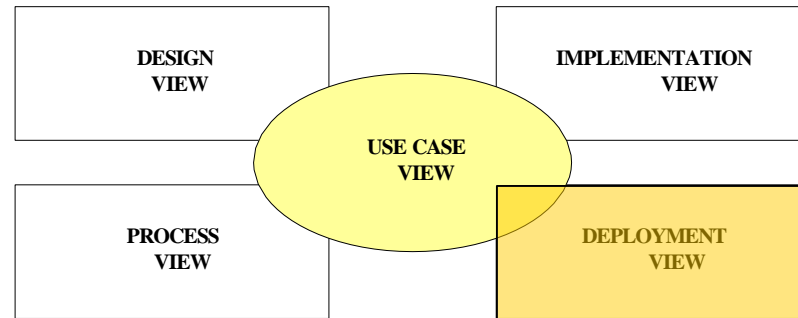
Process View

- La “process view” serve per analizzare gli aspetti non funzionali del sistema, e consiste nell’individuare i processi
 - Lo scopo è un utilizzo efficiente delle risorse
 - poter stabilire l’esecuzione parallela di determinati oggetti
 - poter gestire correttamente eventuali eventi asincroni



Deployment View

- La “deployment view” mostra l’architettura fisica del sistema e l’ubicazione delle componenti software all’interno della struttura stessa



Diagrammi UML

*• I diagrammi di UML sono dei grafici
che visualizzano una particolare proiezione del sistema analizzato
da una specifica prospettiva*

- Use Case Diagram (Diagramma dei casi d'uso)
 - Class Diagram (Diagramma delle classi)
 - Object Diagram (Diagramma degli oggetti)
 - Sequence Diagram (Diagramma di sequenza)
- Collaboration Diagram (Diagramma di collaborazione)
 - Statechart diagram (diagramma degli stati)
 - Activity Diagram (Diagramma delle attività)
- Component diagram (diagramma dei componenti)
- Deployment diagram (diagrammi di dispiegamento)

Casi d'uso

- I casi d'uso costituiscono un ottimo strumento per ottenere una visione d'insieme del sistema che si stà analizzando
 - Sono importanti nelle prime fasi del progetto
- Rappresentano una vista esterna (utilizzatore) del sistema e sono finalizzati a modellare il dialogo tra utilizzatore e sistema:
 - descrivono l'interazione tra attori e sistema, non la "logica interna" della funzione
 - sono espressi in forma testuale, comprensibile anche per i non "addetti ai lavori"
 - possono essere definiti a livelli diversi (sistema o parti del sistema)
 - rappresentano le modalità di utilizzo del sistema da parte di uno o più utilizzatori (attori)
- Ragionare sui casi d'uso aiuta a scoprire i *requisiti* funzionali:
 - i casi d'uso possono essere un valido ausilio nel dialogo con l'utente ed in generale con esponenti non tecnici del progetto

Diagrammi dei casi d'uso

- I diagrammi dei casi d'uso rappresentano i casi d'uso stessi, gli attori e le associazioni che li legano
- Rappresentano le modalità di utilizzo del sistema da parte di uno o più utilizzatori che vengono definiti attori
- I singoli use case non descrivono la “logica interna” delle funzioni ma si occupano di descrivere le iterazioni tra sistema e attori
 - L'attore è un'entità esterna al sistema che fornisce lo stimolo a cui il sistema risponde



- L'attore è un utilizzatore del sistema (essere umano oppure altro sistema) che interagisce con i casi d'uso
- Rappresenta un ruolo di un oggetto oppure un oggetto esterno al sistema che interagisce con lo stesso come parte di un unità di lavoro (work unit) a realizzare un use case
- Un oggetto fisico (o classe) può giocare più ruoli differenti ed essere modellato da diversi attori
- Esempi di attore per un sistema di prenotazioni:
agente di viaggi, addetto check-in, ecc.
- E' bene ricordare nel caso di attori umani che il nome di un attore identifica il ruolo che l'attore svolge in relazione al sistema, non il suo incarico.
 - Esempio: un impiegato (incarico) gestisce ed archivia un documento svolgendo il ruolo di protocollatore

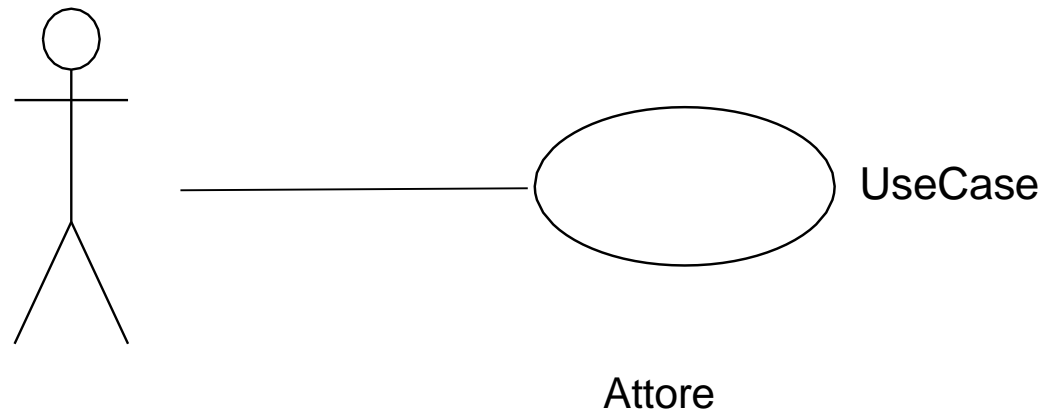
Use case



UseCase

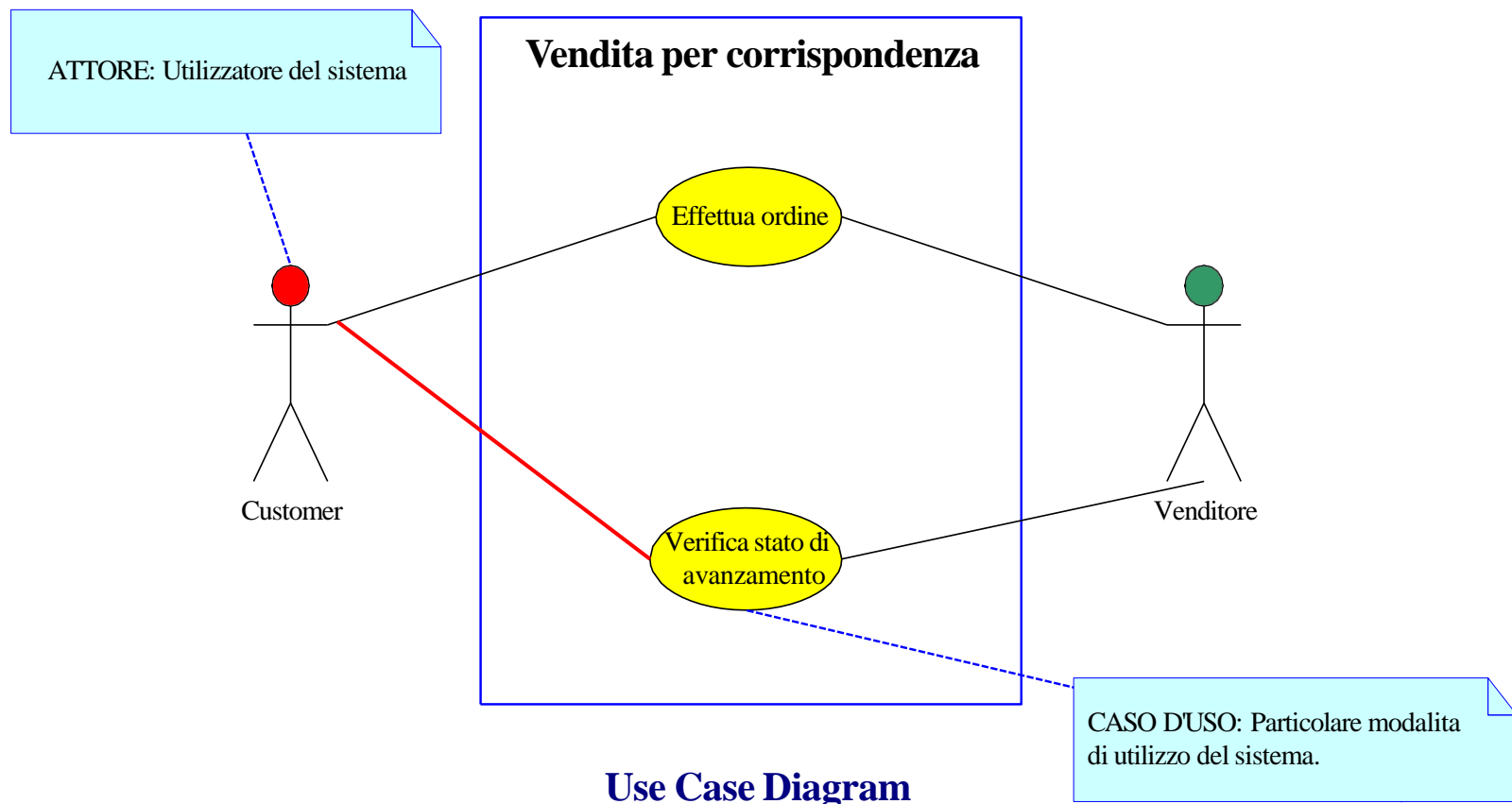
- Il caso d'uso è una particolare modalità di utilizzo del sistema
 - Rappresenta alcune funzioni visibili all'attore
 - Raggiunge alcuni obiettivi per l'attore,
in concreto può essere una scrittura, lettura, modifica di informazioni
- Esempio per un sistema di prenotazioni: checking per un volo, assegnazione di un posto, ecc.

Relazione tra attore e use case



- La relazione indica l'esistenza di un'associazione tra un attore ed un caso d'uso
- *Quindi una particolare persona (o sistema) che si trova in un certo ruolo comunica con specifiche istanze del caso d'uso, partecipando agli eventi rappresentati dal caso*
- In concreto poi il caso d'uso è realizzato come una funzione software utilizzata dagli attori trattando (inserendo o ricevendo) informazioni
 - Nel diagramma gli attori sono collegati ai casi d'uso con i quali interagiscono attraverso una linea che rappresenta la relazione tra loro esistente

Esempio di caso d'uso



Specifica comportamentale

- Ogni caso d'uso costituisce una sequenza di attività che generano un risultato per l'attore che con esso interagisce
- La sequenza di attività viene descritta in una specifica comportamentale
 - può consistere in un diagramma di sequenza, o di collaborazione, o di stato, oppure in istruzioni di un linguaggio di programmazione
- Normalmente viene prodotta una specifica informale nella forma di descrizione del caso d'uso
- La descrizione dei casi d'uso non ha una sintassi formale, a differenza di altri diagrammi UML con precise regole sintattiche, che rendono possibile la simulazione del sistema e la conversione del modello in un programma (ad es. C++ oppure Java)

Descrizione del caso d'uso

- La scelta delle modalità di scrittura della descrizione è demandata al progettista
 - normalmente in un certo contesto (ad es. software house) vi sono delle regole alle quali attenersi
- I due approcci più diffusi per la descrizione:
 - uno o più paragrafi che descrivono la sequenza di attività che si verifica nel caso d'uso
 - elencare, su due colonne, le attività compiute dall'attore e le risposte date dal sistema a tali attività

Descrizione del caso d'uso

- Esempio di scheda UC

CASO D'USO:	Nome:	Data creazione:	
		Versione:	1.001
		Data revisione:	
Descrizione:			
Priorità:			
Durata:			
Punto di estensione:			
Estende:			
Use Case inclusi			
Attore primario:			
Attori secondari:			
Precondizioni:			
Postcondizioni:			
Innesco:			
Scenario principale:			
Scenario alternativo			
Scenario di Errore			

Note:

Riferimento:

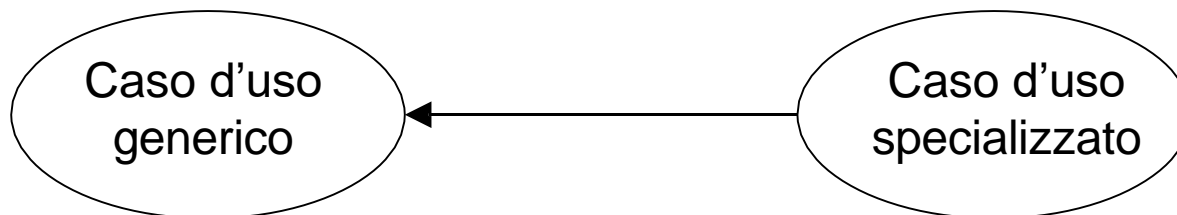
Ad uso esclusivo degli allievi del corso Data Engineer di Generation Italy. Vietata la diffusione all'esterno.

Altri tipi di associazioni e relazioni

- Altri tipi di associazioni e relazioni rappresentabili nel diagramma dei casi d'uso:
 - generalizzazione tra casi d'uso
 - generalizzazione tra attori
 - relazione di inclusione (include) tra casi d'uso
 - relazione di estensione (extend) tra casi d'uso

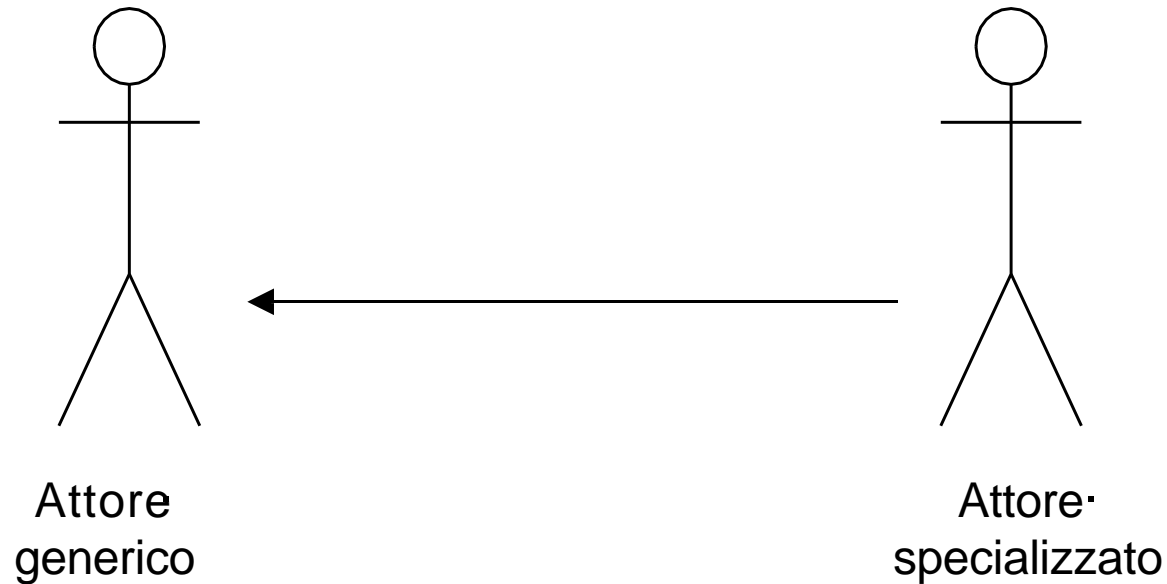
Generalizzazione tra casi d'uso

- Può esistere più di una versione di un caso d'uso, aventi ognuna alcune azioni in comune ed altre uniche per ciascun caso
- Nel diagramma l'associazione di generalizzazione viene indicata da una freccia puntata verso il caso d'uso più generale
 - Il caso d'uso specializzato eredita parte delle funzionalità dal caso d'uso generico



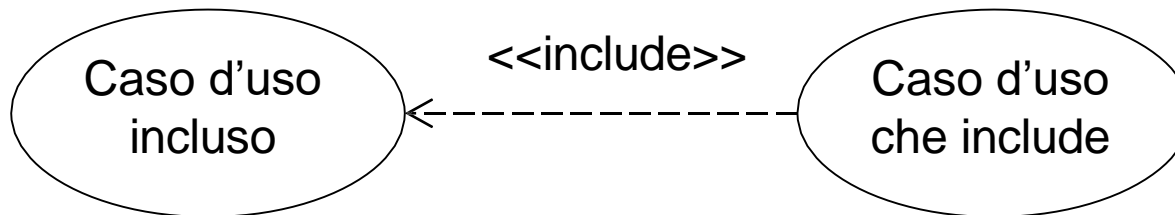
Generalizzazione tra attori

- L'attore specializzato eredita parte delle caratteristiche dell'attore generico



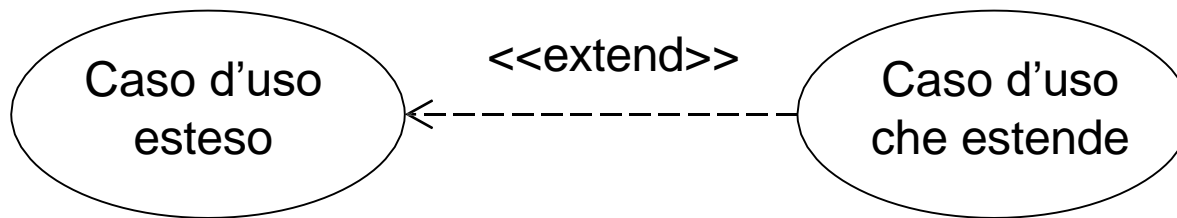
Relazione di inclusione (include) tra casi d'uso

- Un caso d'uso comprende le funzionalità di un altro caso



Relazione di estensione (extend) tra casi d'uso

- Un caso d'uso può essere esteso nella sua funzionalità ad un altro caso



Come produrre casi d'uso

- Nelle prime fase di progetto il progettista/analista lavora tipicamente su appunti e trascrizioni da interviste
- Di norma il processo di analisi e definizione dei casi d'uso avviene per fasi
 - Definire attori e casi d'usotipiche domande alle quali dare risposta:
 - *Chi sono le persone che utilizzeranno questo sistema per inserire informazioni?*
 - *Chi sono i destinatari delle informazioni fornite dal sistema?*
 - *Quali altri sistemi interagiscono con questo?*
 - Organizzare secondo ordine di priorità i casi d'uso
 - I casi d'uso più importanti vanno sviluppati prima
 - Sviluppare ciascun caso d'uso (secondo priorità)
- Generare la specifica dettagliata di ogni caso d'uso (analista del sistema)
 - Strutturare il modello dei casi
- Aggiunta della struttura al diagramma, attraverso generalizzazione, inclusione, estensione ed attraverso il raggruppamento dei casi in “package”

Resources

- Several articles on UML by Scott Ambler - <http://www.agilemodeling.com/artifacts/>
- UML quick reference by Allen Holub - <http://www.holub.com/goodies/uml/>
- Cetus links on UML - http://www.cetus-links.org/oo_uml.html