# Analisi dei casi d'uso (Elementi)

## *Synthesizing on [Architectural and] Use-case Analysis*

In quanto segue, intendiamo presentare una vista di insieme su molti degli aspetti rilevanti fin qui trattati nei corsi di ISPW e LAS.

# Analisi dei casi d'uso (Elementi)

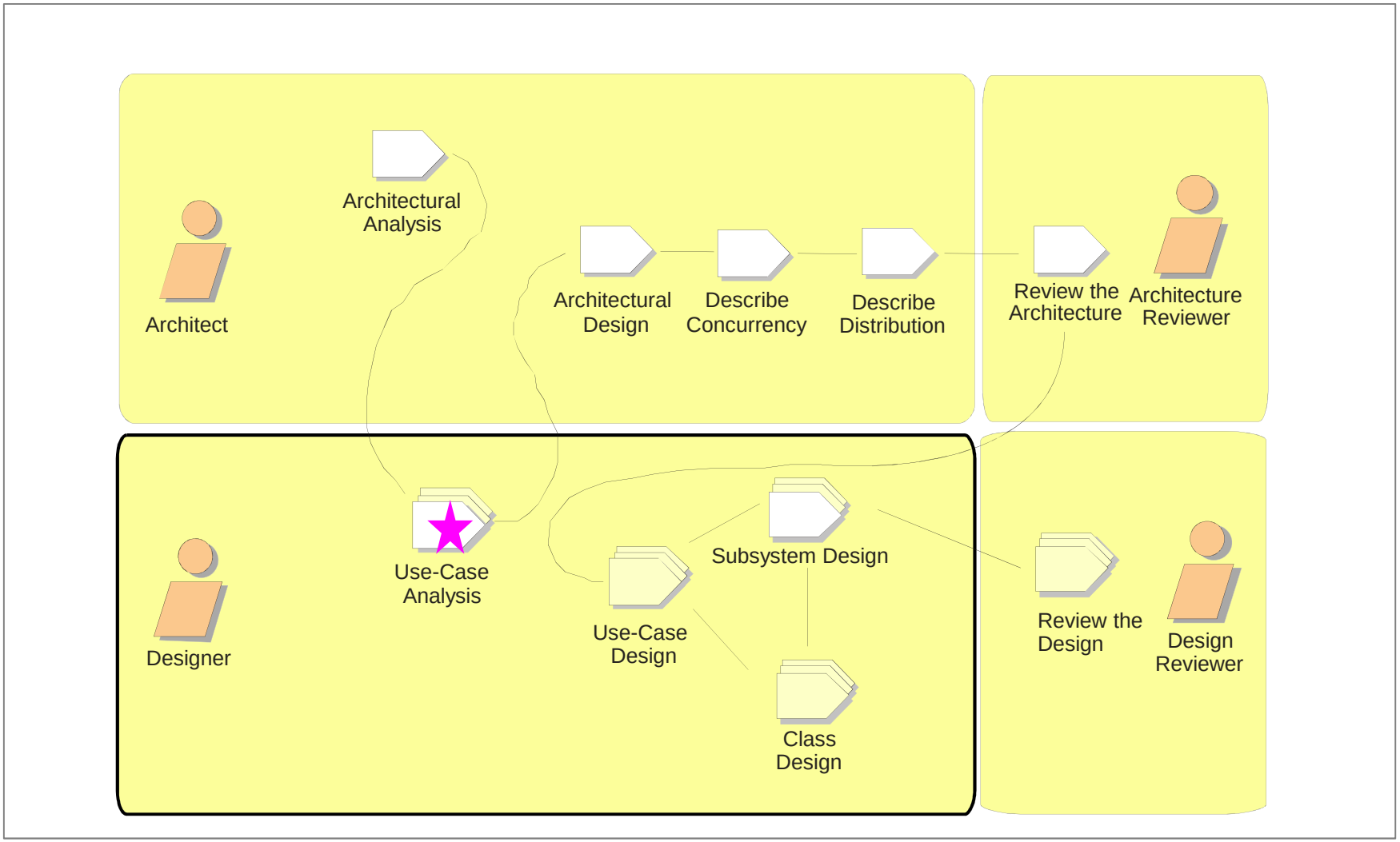
Parte I di VI

# Dai requisiti a classi *Entity*, a *pattern* e *framework*

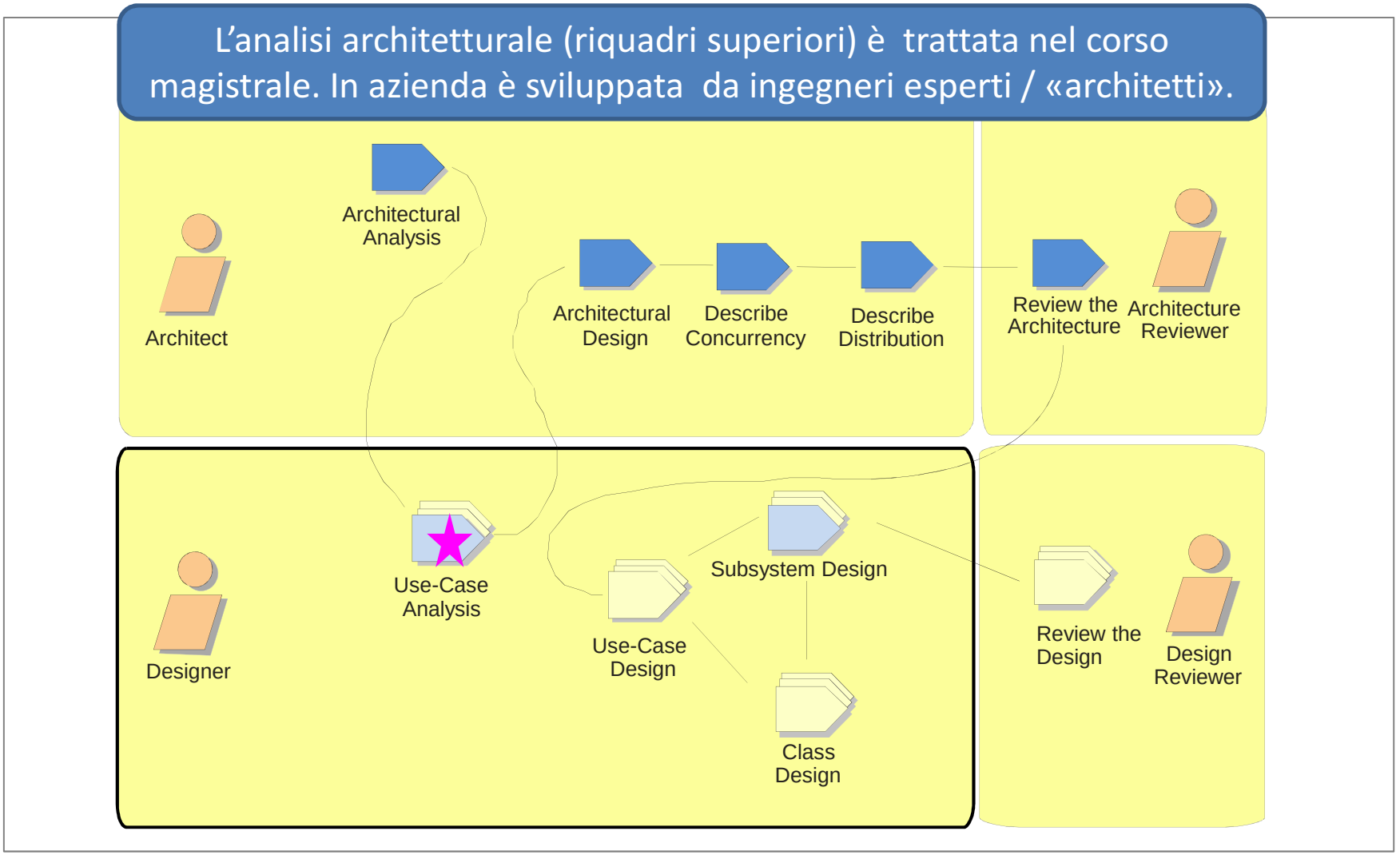
***Credits***
Note basate su materiale Rational® (oggi brand di IBM), come distribuito dall'ing. Giuseppe Calavaro, PhD, agli studenti di Ing. Informatica di Roma "Tor Vergata"

**Queste trasparenze sono intese esclusivamente a supportare i corsi di ISPW & LAS e non a sostituire lezioni e libri.**

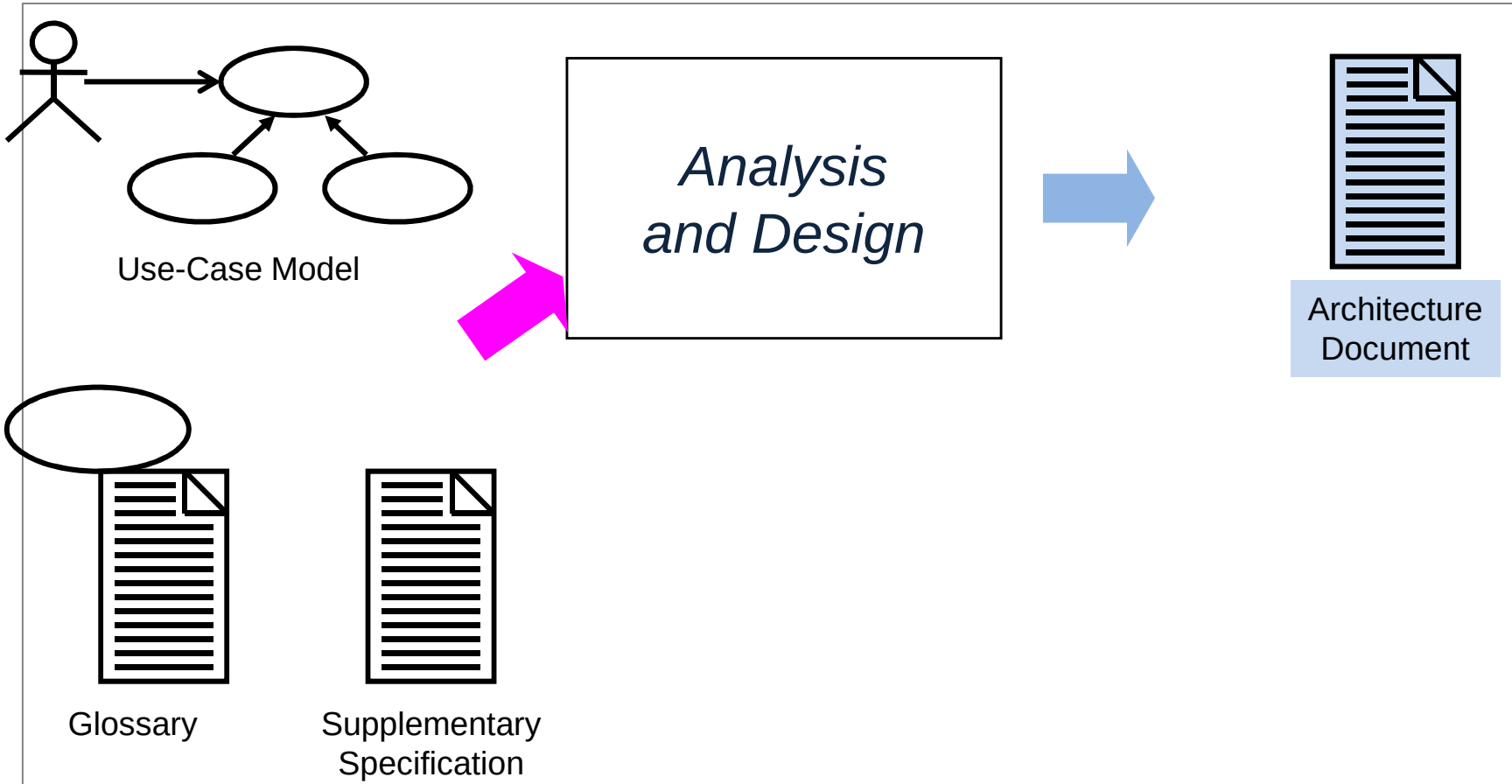
# Analisi architetturale e dei casi d'uso



# Analisi architetturale e dei casi d'uso



# Analisi architetturale



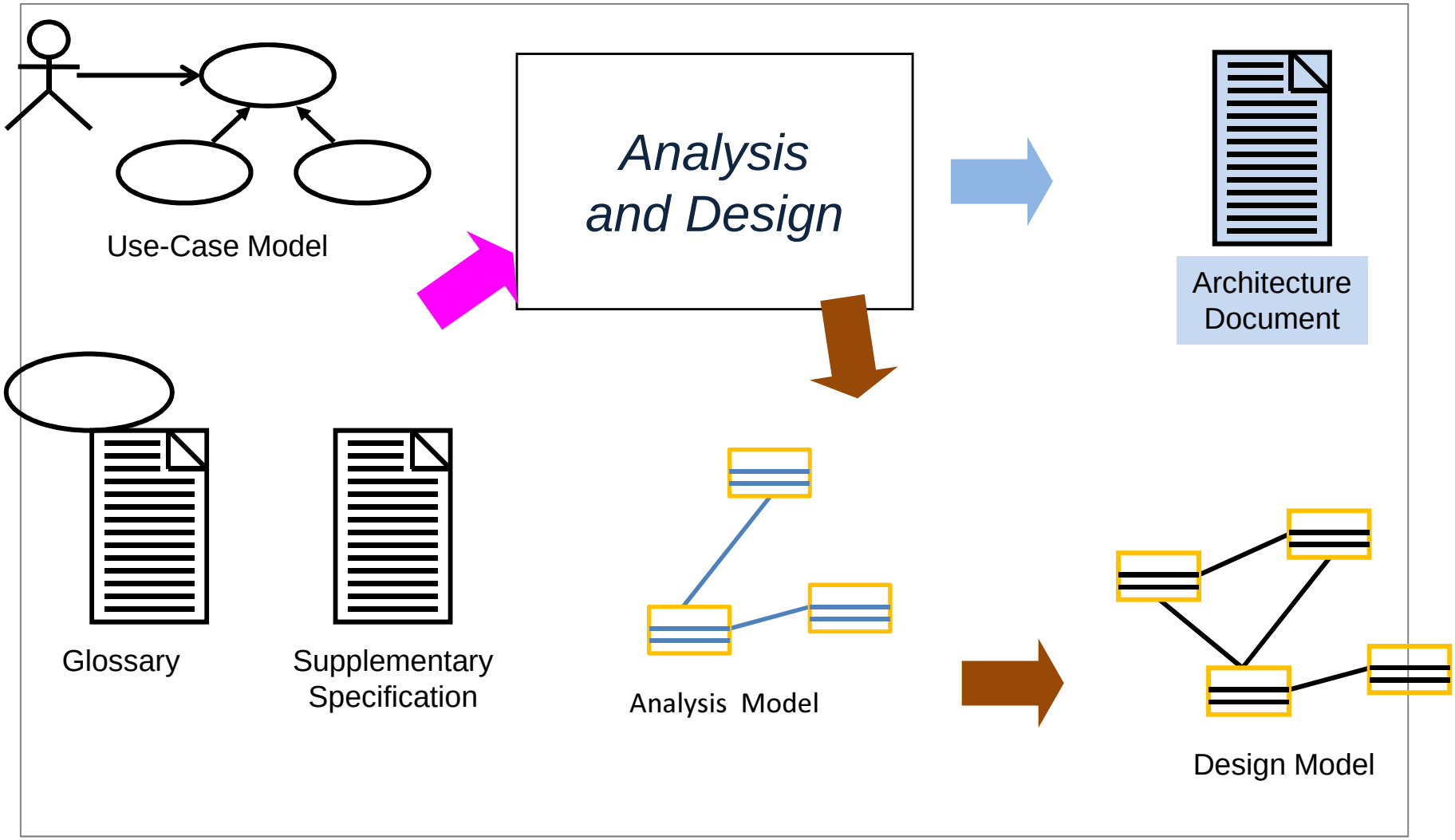
# Esempi di architetture di sistema software (applicativo)

- Stand alone
- Client-Server
- Web-Application
- Service Oriented
- …

# Dai requisiti all'analisi al progetto software

1. Come, nelle precedenti lezioni, vi si è detto di procedere quando si dispone, come risultato della prima iterazione, di Documenti dei Requisiti e Documento di Architettura?
2. Come vi si è detto di procedere per tradurre tali documenti in un *progetto*: prima *concettuale*, infine *esecutivo* (o di *dettaglio*)?
3. Come vi si è detto di procedere passare dall'uno all'altro progetto quasi senza soluzione di continuità?

# Vista d'insieme di analisi e progettazione



# Identificare le astrazioni chiave

1. Definire le classi preliminari di analisi.

# Esempio (non esaustivo) di possibili astrazioni chiave per il progetto del corso

# Identificare le astrazioni chiave

1. Definire le classi preliminari di analisi.

*Sorgenti ?*

- Conoscenza di domino *(Domain knowledge)*
- Requisiti *(Requirements)*
- Glossario *(Glossary)*
- Modello di dominio *(Domain Model) o Modello di business (Business Model),* se esiste
  - *Domain Model:: "Establishes the context of the system".*
  - *Business Model:: "Establishes an abstraction of the organization" [G. Booch]*

# Ci sono ulteriori astrazioni chiave?

- Certamente, ma non ambiamo a trovarle tutte in un colpo solo. Troviamo quelle che ci servono, sono **sufficienti, per cominciare**, così anche evitando che più *team* o diversi membri di un team lavorino su oggetti di base differenti.

# Oltre alle astrazioni chiave, ci sono ulteriori classi di analisi?

Ci si sono già poste le seguenti domande:

- Avremmo, come ingegneri del software, bisogno di classi di analisi di tipologia ulteriore (oltre alle astrazioni chiave)?
- Quali dovrebbero essere tali tipi?

# Ci sono ulteriori classi di analisi?

2. Come abbiamo deciso di rispondere a tali ultime domande?

   Classi di *Boundary* e di *Control* oltre che *Entity*.

   Posponiamo l'approfondimento in merito.

*Come dovremmo procedere per completare l'analisi?*

# Modellare le relazioni fra le classi di analisi

3. Definire le relazioni fra le classi di analisi.

*Che tipi di relazioni sono queste?*

# Modellare le relazioni fra le classi di analisi

4. Modellare le classi di analisi e le relative relazioni tramite diagrammi delle classi.

   *Includere brevi descrizioni delle classi di analisi*

# Modellare le relazioni fra le classi di analisi

5. Mappare le classi di analisi sui necessari «meccanismi» di analisi.

   *Includere (o se programmatori pretendere) brevi descrizioni delle classi di analisi e dei relativi «meccanismi».*

Meccanismi

# Esempi di meccanismi di analisi

- Persistency
- Communication
- Message routing
- Distribution
- Transaction management
- Process control and synchronization (resource contention)
- Information exchange, format conversion
- Security
- Error detection / handling / reporting
- Redundancy
- Legacy Interface

# Esempi di meccanismi che saranno trattati nel corso

- Persistency
- 
- 
- 
- 
- Process control and synchronization (resource contention)
- Information exchange, format conversion
- 
- Error detection / handling / reporting
- 
- 

# Dai requisiti all'analisi al progetto software

Andiamo agli altri due punti dei tre dai quali siamo partiti, tuttora lasciati in sospeso.

1. Come, nelle precedenti lezioni, vi si è detto di procedere quando si dispone, come risultato della prima iterazione, di Documenti dei Requisiti e Documento di Architettura?
2. Come vi si è detto di procedere per tradurre i requisiti e il documento di architettura in un *progetto*: prima *concettuale*, infine *esecutivo* (o di *dettaglio*)?

- Quali altre classi o gruppi o strutture più o meno definitive di classi vi si è indicato di fare intervenire?

3. Come passare dall'uno all'altro progetto quasi senza soluzione di continuità?

# Pattern e Framework

❖**Pattern**

- [A description that includes ] A common solution to a common problem in a context.

*// Vedere definizione estesa in precedenti lezioni (24 .. 26).*

❖**Framework**

- Defines the general approach to solving a problem
- Skeletal solution, whose details may be analysis/design patterns

# Esempi di pattern architetturali

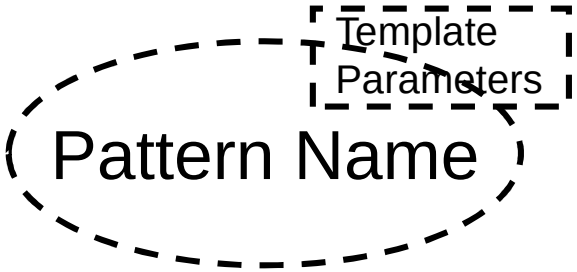
- Layers
- Boundary-Control-Entity
  - Model-View-Controller (MVC)
  - Model-View-Presentation (MVP)
- Pipes and filters
- Blackboard
- …

# Pattern progettuali

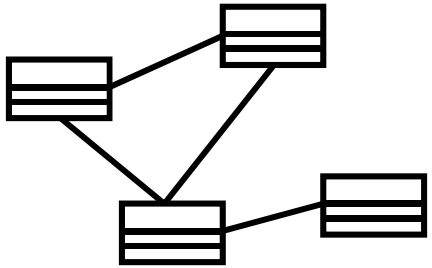
- A design pattern [includes] a solution to a common design problem

  *// Vedere definizione estesa data in precedenti lezioni (24 .. 26).*

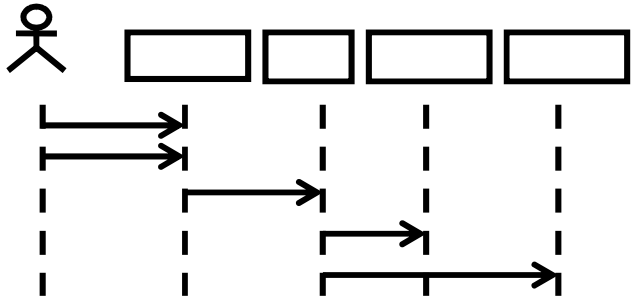
  - Describes a common design problem
  - Describes the solution to the problem
  - Discusses the results and trade-offs of applying the pattern
- Design patterns provide the capability to reuse successful designs




*Parameterized collaboration*



*Structural Aspect*



*Behavioral Aspect*

# Esempi di pattern progettuali (GoF)

| | | Purpose | | |
|---|---|---|---|---|
| | | **Creational** | **Structural** | **Behavioral** |
| **Scope** | **Class** | Factory Method (107) | Adapter (139) | Interpreter (243)<br>Template Method (325) |
| | **Object** | Abstract Factory (87)<br>Builder (97)<br>Prototype (117)<br>Singleton (127) | Adapter (139)<br>Bridge (151)<br>Composite (163)<br>Decorator (175)<br>Facade (185)<br>Proxy (207) | Chain of Responsibility (223)<br>Command (233)<br>Iterator (257)<br>Mediator (273)<br>Memento (283)<br>Flyweight (195)<br>Observer (293)<br>State (305)<br>Strategy (315)<br>Visitor (331) |