Object Oriented Analysis - OOA

- La **fase di OOA** definisce, secondo un approccio ad oggetti, COSA un prodotto software deve fare (mentre la fase di OOD definisce, sempre secondo un approccio ad oggetti, COME un prodotto software deve fare quanto specificato in fase di OOA)
- OOA e OOD devono fornire, ciascuno dal proprio punto di vista, una rappresentazione corretta, completa e consistente:
  - degli aspetti statici e strutturali relativi ai dati (*modello dei dati*)
  - degli aspetti funzionali del sistema (*modello comportamentale*)
  - degli aspetti di "controllo" e di come le funzioni del modello comportamentale modificano i dati introdotti nel modello dei dati (*modello dinamico*)

Metodi di OOA

- Un metodo di OOA definisce l'insieme di procedure, tecniche e strumenti per un approccio sistematico alla gestione e allo sviluppo della fase di OOA
- L'input di un metodo di OOA è costituito dall'*insieme dei requisiti utente* (contenuti nel documento di analisi dei requisiti)
- L'output di un metodo di OOA è costituito dall'*insieme dei modelli del sistema* che definiscono la specifica del prodotto software (e che sono anch'essi contenuti nel documento di analisi dei requisiti)
- I metodi di OOA fanno principalmente uso di notazioni visuali (diagrammi), ma possono essere affiancati da metodi tradizionali per la definizione di requisiti di sistema di tipo testuale (in linguaggio naturale strutturato)
- Lo sviluppo dei modelli di OOA non è un processo sequenziale (prima modello dei dati, poi modello comportamentale, infine modello dinamico)
- La costruzione dei modelli avviene in parallelo, e ciascun modello fornisce informazioni utili per gli altri modelli
- I metodi di OOA fanno uso di un approccio iterativo, con aggiunta di dettagli per raffinamenti successivi (iterazioni)

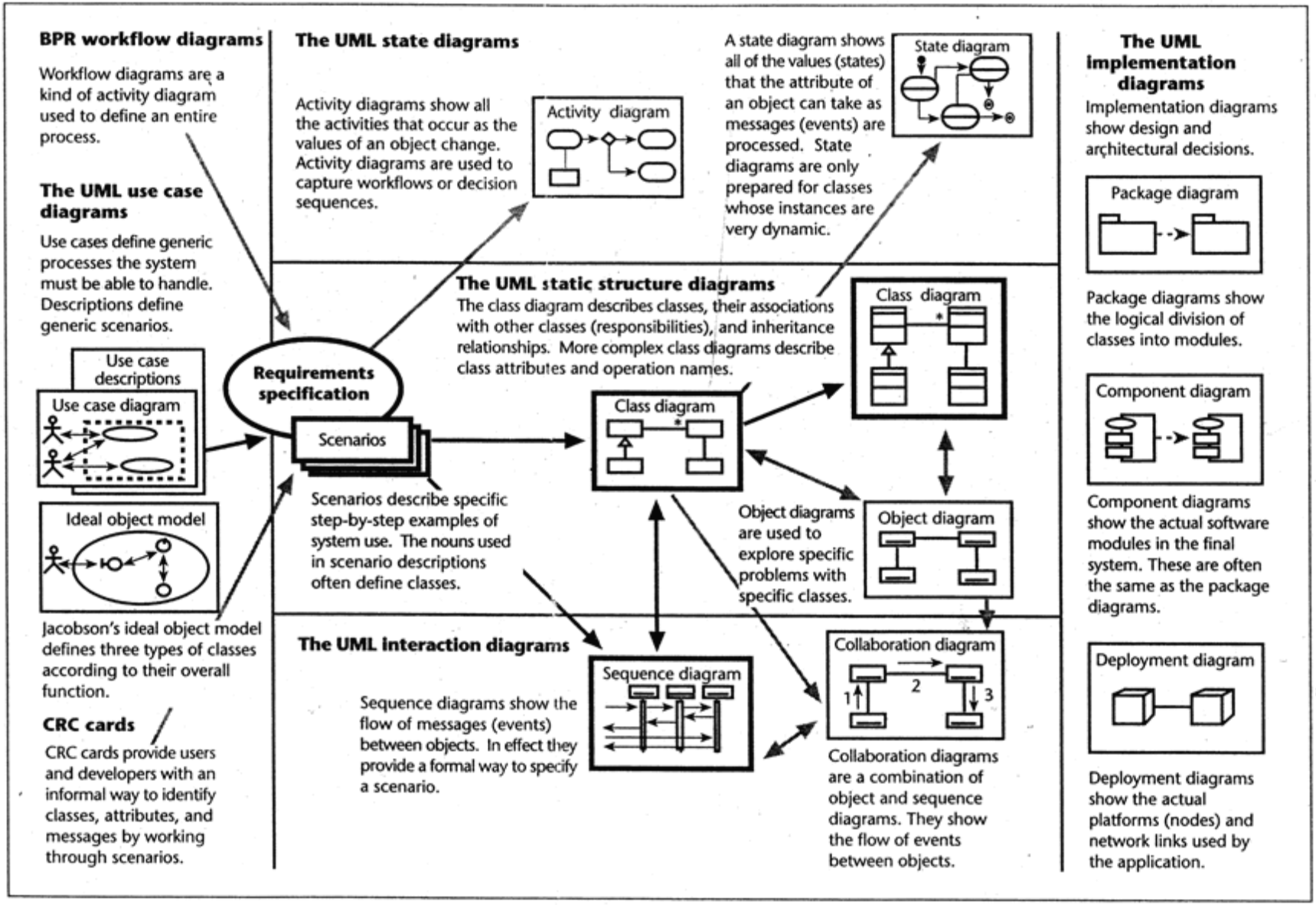
Alcuni metodi di OOA (e OOD)

- Catalysis: metodo OO particolarmente indicato per lo sviluppo di sistemi software a componenti distribuiti.
- Objectory: metodo ideato da I. Jacobson che fonda lo sviluppo di prodotti software ad oggetti sull'individuazione dei casi d'uso utente (*use case driven*).
- Shlaer/Mellor: metodo OO particolarmente indicato per lo sviluppo di sistemi software *real-time*.
- OMT (Object Modeling Technique): metodo sviluppato da J. Rumbaugh basato su tecniche di modellazione del software iterative. Pone in particolare risalto la *fase di OOA*.
- Booch: metodo basato su tecniche di modellazione del software iterative. Pone in particolare risalto la *fase di OOD*.
- Fusion: metodo sviluppato dalla HP a metà degli anni novanta. Rappresenta il primo tentativo di standardizzazione per lo sviluppo di software orientato agli oggetti. Si basa sulla fusione dei metodi OMT e Booch.

Notazioni per OOA (e OOD)

- Ciascun metodo di OOA (e OOD) fa uso di una propria notazione per la rappresentazione dei modelli del sistema
- Al fine di unificare le notazioni per i metodi di OOA e OOD è stato introdotto il linguaggio **UML** (**Unified Modeling Language**), adottato nel 1997 come standard OMG (Object Management Group)
- UML è un linguaggio standard per la descrizione di sistemi software (orientati agli oggetti). Si compone di *nove formalismi di base* (diagrammi con semantica e notazione data) e di un insieme di *estensioni*.
- UML è un linguaggio di descrizione, non è un metodo né definisce un processo
- *Unified Software Development Process* (in breve *Unified Process*) è un tentativo di standardizzazione di processo di sviluppo di sistemi orientati agli oggetti basato sull'uso di UML.

Diagrammi UML



Formalismi UML

I *nove formalismi di base* dello UML sono:

1. Use case diagram

   evidenziano la modalità (caso d'uso) con cui gli utenti (attori) utilizzano il sistema. Possono essere usati come supporto per la definizione dei requisiti utente.

2. Class diagram

   consentono di rappresentare le classi con le relative proprietà (attributi, operazioni) e le associazioni che le legano.

3. State diagram

   rappresentano il comportamento dinamico dei singoli oggetti di una classe in termini di stati possibili e transizioni di stato per effetto di eventi.

4. Activity diagram

   sono particolari state diagram, in cui gli stati rappresentati rappresentano azioni in corso di esecuzione. Sono particolarmente indicati per la produzione di modelli di work-flow.

Formalismi UML (2)

5. Sequence diagram

   evidenziano le interazioni (messaggi) che oggetti di classi diverse si scambiano nell'ambito di un determinato caso d'uso, ordinate in sequenza temporale. A differenza dei diagrammi di collaborazione, non evidenziano le relazioni tra oggetti.

6. Collaboration diagram

   descrivono le interazioni (messaggi) tra oggetti diversi, evidenziando le relazioni esistenti tra le singole istanze.

7. Object diagram

   permettono di rappresentare gli oggetti e le relazioni tra essi nell'ambito di un determinato caso d'uso.

8. Component diagram

   evidenziano la strutturazione e le dipendenze esistenti tra componenti software.

9. Deployment diagram

   evidenziano le configurazioni dei nodi elaborativi di un sistema real-time ed i componenti, processi ed oggetti assegnati a tali nodi.

Modello dei dati

- Rappresenta da un punto di vista statico e strutturale l'organizzazione logica dei dati da elaborare
- Le strutture dati sono definite mediante lo stato degli oggetti, che viene determinato dal valore assegnato ad attributi e associazioni
- Il modello dei dati viene specificato mediante il formalismo dei *class diagram* che permette di definire:
  - classi
  - attributi di ciascuna classe
  - operazioni di ciascuna classe
  - associazioni tra classi
- Il modello dei dati è di fondamentale importanza, visto che, secondo l'approccio ad oggetti, un sistema software è costituito da un insieme di oggetti (classificati) che collaborano

Modello dei dati (2)

- Il modello dei dati viene costruito in modo *iterativo* ed *incrementale*
- Si tratta di un processo creativo, in cui giocano un ruolo importante sia l'esperienza dell'analista che la comprensione del dominio applicativo
- Durante la fase iniziale di costruzione del modello dei dati occorre concentrarsi sulle cosiddette ***entity classes***, ovvero quelle classi che definiscono il dominio applicativo e che sono rilevanti per il sistema
- Le *control classes* (che gestiscono la "logica" del sistema) e *boundary classes* (che rappresentano l'interfaccia utente) vengono introdotte successivamente, usando le informazioni del *modello comportamentale*
- Le operazioni di ciascuna classe vengono identificate a partire dal *modello comportamentale*, per cui vengono inizialmente trascurate

Approcci per l'identificazione delle classi

- *Noun phrase*
- *Common class patterns*
- *Use case driven*
- *CRC*
- *Mixed*

Approccio *noun phrase*

- Una frase nominale (*noun phrase*) è una frase in cui il sostantivo ha una prevalenza sulla parte verbale (sono frasi di tipo *assertivo*)
- I sostantivi delle frasi nominali usate per la stesura dei requisiti utente sono considerati ***candidate classes***
- La lista delle *candidate classes* viene suddivisa in tre gruppi:
  - *Irrelevant* (non appartengono al dominio applicativo e quindi possono essere scartate)
  - *Relevant* (evidenziano caratteristiche di *entity classes*)
  - *Fuzzy* (non si hanno sufficienti informazioni per classificarle come *relevant* o *irrelevant*, vanno analizzate successivamente)
- Si assume che l'insieme dei requisiti utente sia completo e corretto

Approccio *common class patterns*

- Basato sulla *teoria della classificazione*
- Le *candidate classes* vengono identificate a partire da gruppi (*pattern*) di classi predefinite:
  - *Concept* (es. *Reservation*)
  - *Events* (es. *Arrival*)
  - *Organization* (es. *AirCompany*)
  - *People* (es. *Passenger*)
  - *Places* (es. *TravelOffice*)
- Non è un approccio sistematico, ma può rappresentare una utile guida
- A differenza dell'approccio *noun phrase*, non si concentra sul documento dei requisiti utente
- Può causare problemi di interpretazione dei nomi delle classi

Approccio *use case driven*

- Si assume che:
  - Siano già stati sviluppati gli *use case diagram* (e possibilmente anche i *sequence diagram* più significativi)
  - Per ogni *use case* sia fornita una descrizione testuale dello scenario di funzionamento
- Simile all'approccio *noun phrase* (si considera l'insieme degli *use case* come insieme dei requisiti utente)
- Si assume che l'insieme degli *use case* sia completo e corretto
- Approccio *function-driven* (o *problem-driven* secondo la terminologia object oriented)

Approccio *CRC*

- L'approccio CRC (Class - Responsibility – Collaborators) è basato su riunioni in cui si fa uso di apposite *card*
- Ciascuna *card* rappresenta una classe, e contiene tre compartimenti, che identificano:
  - Il nome della classe
  - Le responsabilità assegnate alla classe
  - Il nome di altre classi che collaborano con la classe
- Le classi vengono identificate analizzando come gli oggetti collaborano per svolgere le funzioni di sistema
- Approccio utile per
  - *Verifica di classi* identificate con altri metodi
  - *Identificazione di attributi e operazioni* di ciascuna classe

CLASS Elevator Controller
RESPONSIBILITY
1. Turn on elevator button
2. Turn off elevator button
3. Turn on fl oor button
4. Turn off fl oor button
5. Open elevator doors
6. Close elevator doors
7. Move elevator one fl oor up
8. Move elevator one fl oor down
COLLABORATION
1. Class **Elevator Button**
2. Class **Floor Button**
3. Class **Elevator**

Approccio *mixed*

- Basato su elementi presenti in ciascuno degli approcci precedenti
- Un possibile scenario potrebbe essere il seguente:
  1. L'insieme iniziale delle classi viene identificato in base all'esperienza dell'analista, facendosi eventualmente guidare dall'approccio *common class patterns*
  2. Altre classi possono essere aggiunte usando sia l'approccio *noun phrase* che l'approccio *use case driven* (se gli *use case diagram* sono disponibili)
  3. Infine l'approccio *CRC* può essere usato per verificare l'insieme delle classi identificate

Linee guida per l'identificazione delle *entity classes*

1. Ogni classe deve avere un ben preciso *statement of purpose*
2. Ogni classe deve prevedere un insieme di istanze (oggetti) – le cosiddette *singleton classes* (per la quali si prevede una singola istanza) non sono di norma classificabili come *entity classes*
3. Ogni classe deve prevedere un insieme di attributi (non un singolo attributo)
4. Distinguere tra elementi che possono essere modellati come classi o come attributi
5. Ogni classe deve prevedere un insieme di operazioni (anche se inizialmente le operazioni vengono trascurate, i servizi che la classe mette a disposizione sono implicitamente derivabili dallo *statement of purpose*)

Casi di studio (*)

A. *University Enrolment*

B. *Video Store*

C. *Contact Management*

D. *Telemarketing*

(*) MACIASZEK, L.A. (2001): *Requirements Analysis and System Design. Developing Information Systems with UML*, Addison Wesley

A. University Enrolment
Problem statement

- The **university** offers
  - Undergraduate and postgraduate degrees
  - To full-time and part-time students
- The **university structure**
  - Divisions containing departments
  - Single division administers each degree
  - Degree may include courses from other divisions
- **University enrolment** system
  - Individually tailored programs of study
  - Prerequisite courses
  - Compulsory courses
  - Restrictions
    - Timetable clashes
    - Maximum class sizes, etc.

A. University Enrolment
Problem statement (2)

- The system is required to
  - Assist in pre-enrolment activities
  - Handle the enrolment procedures
- **Pre-enrolment activities**
  - Mail-outs of
    - Last semester's examination grades to students
    - Enrolment instructions
- **During enrolment**
  - Accept students' proposed programs of study
  - Validate for prerequisites, timetable clashes, class sizes, special approvals, etc.
- Resolutions to some of the problems may require consultation with academic advisers or academics in charge of course offerings

B. Video Store
Problem statement

- The **video store**
  - Rentals of video tapes and disks to customers
  - All video tapes and disks bar-coded
  - Customer membership also be bar-coded.
- Existing customers can place reservations on videos to be collected at specific date
- Answering customer enquiries, including enquiries about movies that the video store does not stock (but may order on request)

C. Contact Management
Problem statement

- The **market research company** with established customer base of organizations that buy market analysis reports
- The company is constantly on the search for new customers
- **Contact management** system
  - Prospective customers
  - Actual customers
  - Past customers
- The new contact management system to be developed internally and be available to all employees in the company, but with varying levels of access
  - Employees of Customer Services Department will take the ownership of the system
- The system to permit flexible scheduling and re-scheduling of contact-related activities so that the employees can successfully collaborate to win new customers and foster existing relationships

D. Telemarketing
Problem statement

- The **charitable society** sells lottery tickets to raise funds
  - **Campaigns** to support currently important charitable causes
  - Past contributors (**supporters**) targeted through telemarketing and/or direct mail-outs
- Rewards (special bonus campaigns)
  - For bulk buying
  - For attracting new contributors
- The society does not randomly target potential supporters by using telephone directories or similar means

D. Telemarketing
Problem statement (2)

- **Telemarketing application**
  - To support up to fifty telemarketers working simultaneously
  - To schedule the phone calls according to pre-specified priorities and other known constraints
  - To dial up the scheduled phone calls
  - To re-schedule unsuccessful connections
  - To arrange other telephone callbacks to supporters
  - To records the conversation outcomes, including ticket orders and any changes to supporter records