- ***Lezione 12***

Esistono alcuni metodi che partono dal modello comportamentale e altri metodi che partono dal modello dei dati.

Noi ci focalizzeremo sul *Modello dei Dati.* Questo 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

Questo modello di dati è un modello 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

Nella fase iniziale di costruzioni del modello dei dati occorre concentrarsi sulle cosiddette *entity classes,* ovvero quelle classi che definiscono il dominio applicativo e che sono rilevanti per lo sviluppo del software. Quello che dovremmo fare per costruire il modello di dati è costruire un *class diagram*, facendo uso della notazione *UML*, nel quale andiamo ad identificare le *classi entity*, specificandole in termini di attributi e associazioni e tralasciando l'identificazione delle operazioni in un secondo momento.

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

In particolare vediamo un insieme di tecniche o approcci che ci aiutano in termine di identificazioni delle classi. Questi approcci sono 4, l'ultimo è una combinazione dei precedenti:

- Noun phrase → questo è l'approccio delle frasi nominali (noun phrase). Dove per frase nominale intendiamo una frase in cui il sostantivo prevale sulla parte verbale (sono frasi di tipo assertivo). I sostantivi delle frasi nominali usate per la stesura dei requisiti utente sono considerati classi candidate (candidate classes). La lista delle classi candidate viene suddivisa in tre gruppi:

- *Irrilevante (irrelevant)* → quindi non appartengono al dominio applicativo e quindi possono essere scartate e non diventare una entity classes
- *Rilevante (Relevant)* → quindi una classe che posso identificare come entity classes
- *Fuzzy* → sono quelle classi delle quali non abbiamo sufficienti informazioni per classificarle come rilevanti o irrilevanti, vanno analizzate successivamente.

N.B: Si assume che l'insieme dei requisiti utente sia completo e corretto.

- *Common class patterns* → questo approccio è basato sulla teoria della classificazione, la quale studia il partizionamento in gruppi significativi detti *pattern.* Si hanno un certo insieme di classi predefinite, possiamo vedere un esempio di software per la prenotazione di biglietti aerei:

  - concetti (es. Reservation)
  - eventi (es. Arrival)
  - oerganizzazioni (es. AirCompany)
  - persone (es. Passenger)
  - posti (es. TravelOffice)

  Non è un approccio sistematico, ma può essere utile come guida. A differenza dell'approccio noun phrase, non si concentra sul documento dei requisiti utente. Può causare problemi di interpretazione dei nomi delle classi.

- *Use case driven* → è un approccio derivato da Jacobs. Nel quale i requisiti utente sono definiti con un *use case diagram* (diagramma dei casi d'uso). 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)

- *CRC* → L'approccio *CRC* (*Class - Responsibility – Collaborators*) è basato su riunioni in cui si fa uso di apposite card. Sono riunioni condotte dal team di analisti software, dove questi usano queste card. Ogni 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. Questo approccio è utile per verifica di classi identificate con altri metodi e per l'identificazione di attributi e operazioni di ciascuna classe.

- Mixed → è una combinazione 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

Le seguenti linee guida ci servono per capire quando una classe può essere definita entity

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)