- ***Lezione 11 (parte due)***

- *Quinto Blocco*

La fase di OOA (Objet Oriented Analisis) definisce cosa un prodotto software deve fare, senza occuparsi del come. Successivamente [secondo modulo] vedremo la fase di OOD (Objet Oriented Desiner).

Queste due fasi devono fornire, ciascuno dal proprio punto di vista, una rappresentazione corretta, completa* e consistente:

- dal punto di vista dei dati → il modello dei dati si occupa degli aspetti statici e strutturali, deve descrivere i dati che dovranno essere gestiti dal software
- dal punto di vista comportamentale → il modello comportamentale si descrive sulle funzioni che dovranno elaborare e processare questi dati
- dal punto di vista dinamico → il modello dinamico si sofferma sugli aspetti di controllo e su come questi dati vengano modificati.

Completa → significa che oltre a considerare tutti i requisiti deve anche rappresentare e specificare il software da ogni punto di vista, quali il modello dei dati, il modello comportamentale e il modello dinamico.

Questi metodi di OOA, definiscono l'insieme di procedure, tecniche, linguaggi e strumenti per un approccio sistematico alla gestione e allo sviluppo della fase di queste attività.

Attività che riceve come input l'insieme dei requisiti utente (contenuti nel documento di analisi dei requisiti) e come output produce l'insieme dei modelli del sistema che definiscono la specifica del prodotto software (e che sono anch'essi contenuti nel documento di analisi dei requisiti)

Questi metodi sfruttano un approccio semi-formale e quindi 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 di questi avviene in parallelo (concorrente), 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)

Non esiste uno o tanti metodi, possiamo analizzare i seguenti:

- *Catalysis:* metodo di analisi e progettazione OOAD particolarmente indicato per lo sviluppo di sistemi software a componenti distribuiti.

- *Objectory:* metodo ideato da I. Jacobson, si basa sull'individuazione dei casi d'uso utente (*use case driven*).
- *Shlaer/Mellor:* metodo 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.

Ogni metodo aveva una sua notazione per la rappresentazione dei modelli del sistema., negli anni si è pensato di tentare di proporre un linguaggio standard. Questo linguaggio standard si chiama UML.

Questo è stato adottato nel 1997 come standard OMG (Object Management Group).

L'origine di UML è legata agli autori di tre metodi visti prima, ovvero:

- *Objectory*
- *OMT*
- *Booch*

Questi autori si sono trovati a lavorare all'interno di un'azienda, chiamata Rational Software. Oggi non esiste più, però è stata un'azienda molto importante. Questi tre autori si sono trovati insieme nei primi anni '90 e hanno combinato i relativi linguaggi di modellazione dei vari metodi e hanno sviluppato questo linguaggio (univoco) di modellazione.

Prima di definirlo come standard è stato necessario seguire una procedura molto complessa. Questo è un processo molto lungo, il tutto nasce con RFP (Request for Propusa) nel quale nasce l'esigenza di creare un linguaggio di modellazione unificato.

Si compone di nove formalismi di base (diagrammi con semantica e notazione data) e di un insieme di estensioni. All'interno di questo standard è stato introdotto un meccanismo standard di estensione, per dare la possibilità di estendere UML mantenendo la compatibilità verso la versione base del linguaggio.

[*N.B:* UML è un linguaggio di descrizione, non è un metodo né definisce un processo]

Negli anni si è pensato di fare un ulteriore tentativo, che prende il nome di *Unified Software Development Process* (in breve *Unified Process*), nel quale si è pensato di standardizzazione di processo di sviluppo di sistemi orientati agli oggetti basato

sull'uso di UML. A volte si fa riferimento a questo con la denominazione di RUP, dove *r* sta per *Rational* poiché è avvenuto anch'esso all'interno di questa azienda.

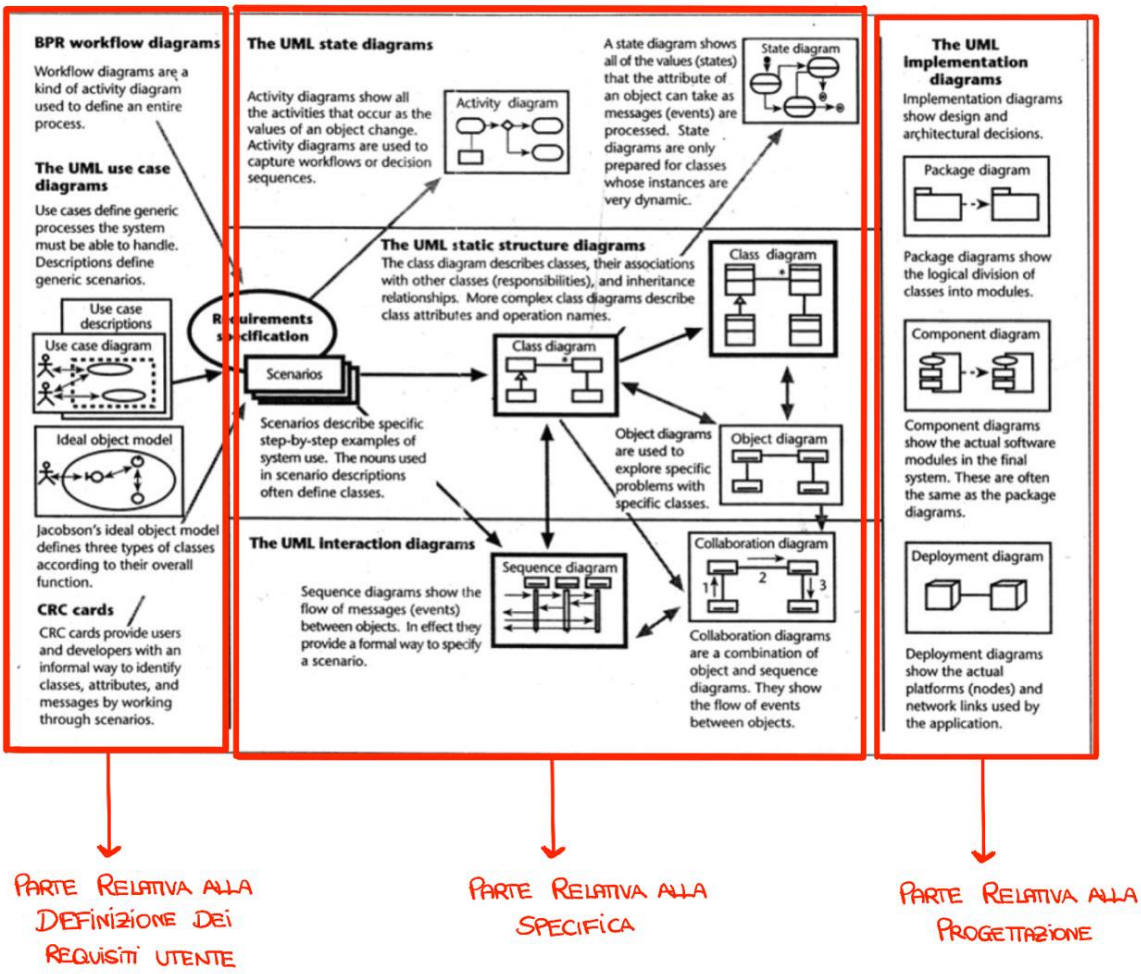

Diagramma UML

I nove formalismi 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.
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.