

1 Lezione del 26-09-25

1.1 Introduzione all'UML

L'**UML** (*Unified Modeling Language*) è una notazione grafica standardizzata, aperta ed estensibile, di modellazione. L'UML è pensato per modellizzare progetti che sfruttano la programmazione ad oggetti (OOP).

Viene detto *unificato* in quanto accompagna il software in tutti i suoi cicli di vita, dalle specifiche all'implementazione. Può essere usato per modellare diversi domini applicativi (dall'embedded alle applicazioni, ecc...), ed è trasparente al linguaggio e alle metodologie usate.

I modelli di UML rappresentano collezioni di **oggetti interagenti**. In particolare sfruttiamo modelli:

- A **struttura statica**, cioè che dettagliano quali oggetti compongono il nostro progetto in maniera statica;
- A **comportamento dinamico**, cioè dettagliamo successivamente come questi oggetti interagiscono fra loro nel tempo.

1.2 Introduzione all'UP

L'**UP** (*Unified Process*) è un processo di ingegnerizzazione software basato su 3 principi fondamentali:

1. Guidato dall'analisi dei *requisiti* e dei *rischi*;
2. Centrato sull'*architettura*, cioè finalizzato alla produzione di un'architettura robusta;
3. *Iterativo ed incrementale*, cioè suddivide il progetto in iterazioni incrementali che arrivano da zero al sistema funzionante.

Ciascuna iterazione è finalizzata a uno o più *sotto-obiettivi*:

- Pianificazione;
- Analisi e progetto;
- Costruzione;
- Integrazione e test;
- Release interna o esterna.

Ogni iterazione genera la cosiddetta *baseline*, cioè la versione da cui partirà la prossima iterazione, e via dicendo. L'*incremento* sarà rappresentato dalla variazione da una baseline alla successiva.

Le fasi vengono implementate seguendo sostanzialmente uno o più fra 5 workflow riassunti dalla sigla **RADIT**:

- **Requirements** (requisiti);
- **Analysis** (analisi);

- **Design** (design);
- **Implementation** (implementazione);
- **Test** (test).

1.2.1 Struttura dell'UP

Il ciclo di vita del progetto si evolve in più iterazioni (ciascuna delle quali comprende i 5 workflow RADIT) . In particolare individuiamo 4 fasi:

1. **Inception** (principio): qui l'obiettivo è far partire il progetto. Dobbiamo quindi stabilire la *fattibilità* del progetto, creare un caso di business (cioè dimostrare la redditività del progetto), catturare le specifiche di base ed individuare i primi rischi critici.

Gli workflow principali saranno requisiti ed analisi.

Vediamo quelle che sono le *milestone* associate a questa fase: vogliamo stabilire un'associazione fra *condizioni* da soddisfare e **deliverable** (*consegnabili*) che possiamo appunto dare come ottenuti. In particolare, potremmo avere:

Condizioni	Deliverable
Le persone coinvolte sono d'accordo sugli obiettivi di progetto	Un documento che riassume i requisiti principali di progetto
Viene tracciata l'architettura generale	Un documento che delinea l'architettura generale
Si crea un primo piano di progetto	Il piano di progetto

2. **Elaboration** (elaborazione): è la fase dove si delinea un'architettura eseguibile, si perfezionano i rischi valutati, si definiscono gli *attributi di qualità*, si cerca di catturare almeno l'80% delle specifiche funzionali, si crea un piano dettagliato per la fase di costruzione e si formula un'offerta per il cliente che comprende risorse, tempo e staff richiesto.

Gli workflow principali includeranno requisiti, analisi e la prima fase di design.

Milestone in questo saranno ad esempio:

Condizioni	Deliverable
Viene creata un'architettura eseguibile	L'architettura eseguibile
L'architettura dimostra di aver individuato i rischi importanti	I modelli UML statico, dinamico e dei casi d'uso
Si crea un piano di progetto realistico e realizzabile	Un piano di progetto aggiornato

3. **Construction** (costruzione): in questo caso si prende l'architettura delineata in fase di elaborazione e si inizia a sviluppare il prodotto software vero e proprio.

Il workflow principale sarà caratterizzato da design e sviluppo, nonché pesante testing.

Le milestone includeranno:

Condizioni	Deliverable
Il prodotto software è sufficientemente stabile	Il prodotto software, documentazione
I committenti sono pronti per l'installazione del software	Manuali, documentazione

4. **Transition** (transizione): questa è la fase dove si risolvono i difetti delle versioni beta e si prepara l'installazione del software nell'infrastruttura dell'utente. Inoltre si realizzano i manuali utente ed eventualmente si fornisce consulenza.

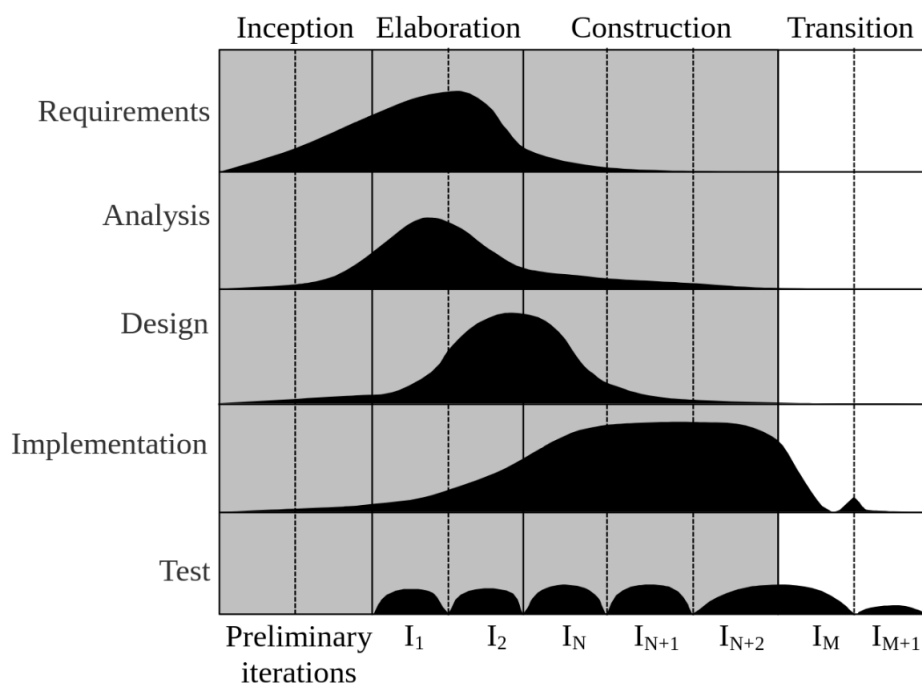
Il workflow comprenderà sviluppo e testing delle ultime funzionalità.

Le milestone saranno ristrette:

Condizioni	Deliverable
Il prodotto è stabile e (perlopiù) privo di bug	Il prodotto software finito

Ciascuna fase corrisponde a una o più iterazioni. Non è detto che lo "sforzo" (*effort*) su ogni workflow sia però lo stesso su ogni workflow nelle diverse fasi: abbiamo infatti dettagliato quali sono gli workflow più indicati per ogni fase.

Possiamo schematizzare lo sforzo sugli workflow per ogni fase con il seguente grafico:



dove per ogni fascia associata agli workflow (Requirements, Analysis, ecc...) l'andamento del carico su quel workflow per ogni iterazione (I₁, ecc...).

Notiamo come ad esempio la fase di testing diventa parte integrante di ogni iterazione praticamente dalla fase di elaborazione, se non per l'inizio dell'iterazione (dove presumibilmente stiamo progettando/implementando quanto sarà testato).

1.3 Workflow requisiti

Il workflow requisiti ha compito di individuare i requisiti del sistema. Questi sono di due tipi:

- **Funzionali:** legati a *cosa* il sistema deve fare;
- **Non funzionali:** legati a *come* il sistema deve funzionare.

Per definire i requisiti in UML possiamo usare un formato molto semplice, del tipo:

```
1 <id> Il <nome del sistema> deve <funzione da realizzare>
```

dove <id> identifica un requisito.

Quando i requisiti diventano molti, è utile raggrupparli per tipologia. 2 o 3 livelli di profondità della gerarchia sono appropriati finché non si lavora con requisiti particolarmente complessi.

Ogni requisito può essere corredato di uno o più *attributi*, cioè coppie chiave/valore associate al requisito stesso.

1.3.1 Analisi delle priorità

L'attributo più comune dei requisiti è la **priorità**. Questa si definisce secondo l'acronimo MoSCoW, cioè:

- **Must have:** requisiti fondamentali per il sistema;
- **Should have:** requisiti importanti che possono (dopo opportuna discussione) essere omessi;
- **Could have:** requisiti opzionali (da realizzare se possibile, cioè se c'è tempo);
- **Want to have:** requisiti che non verranno realizzati adesso, ma al massimo in successive release.

1.3.2 Individuazione dei requisiti

I requisiti sono generati dal contesto di sistema che si vuole modellare, comprensivo di:

- Gli utenti del sistema;
- Le altre persone coinvolte (installatori, ecc...);
- I sistemi con cui il sistema deve interagire;
- I requisiti hardware del sistema e altri vincoli tecnici;
- Vincoli legali e regolamenti;
- L'obiettivo di business nostro e del cliente.

L'individuazione dei requisiti genera solitamente un documento di visione d'insieme, scritto in linguaggio naturale, che delinea i requisiti realizzabili del progetto.

Un processo che possiamo usare è quello di *deduzione* dei requisiti, tecnica dove si cerca di estrarre i requisiti dalle persone coinvolte nel progetto.

Altre metodologie sono le *interviste*, i *questionari* e i *gruppi di lavoro*.

1.3.3 Modellizzazione casi d'uso

La **modellizzazione dei casi d'uso** fa parte dell'ingegnerizzazione dei requisiti, e consiste nell'individuare come gli *attori* (sostanzialmente gli utenti e gli altri elementi coinvolti nell'uso del sistema) interagiscono con questo, all'interno del *soggetto* (il dominio di operazione del sistema).



Questo processo si svolge nel modo seguente:

- Identificare un *confine* candidato del sistema, cioè il dominio di operazione del sistema stesso. Identificare il confine del sistema significa capire cosa il sistema è e cosa non è. Questo aiuta nella definizione delle specifiche funzionali.

In UML i confini del sistema sono chiamati **soggetto**;

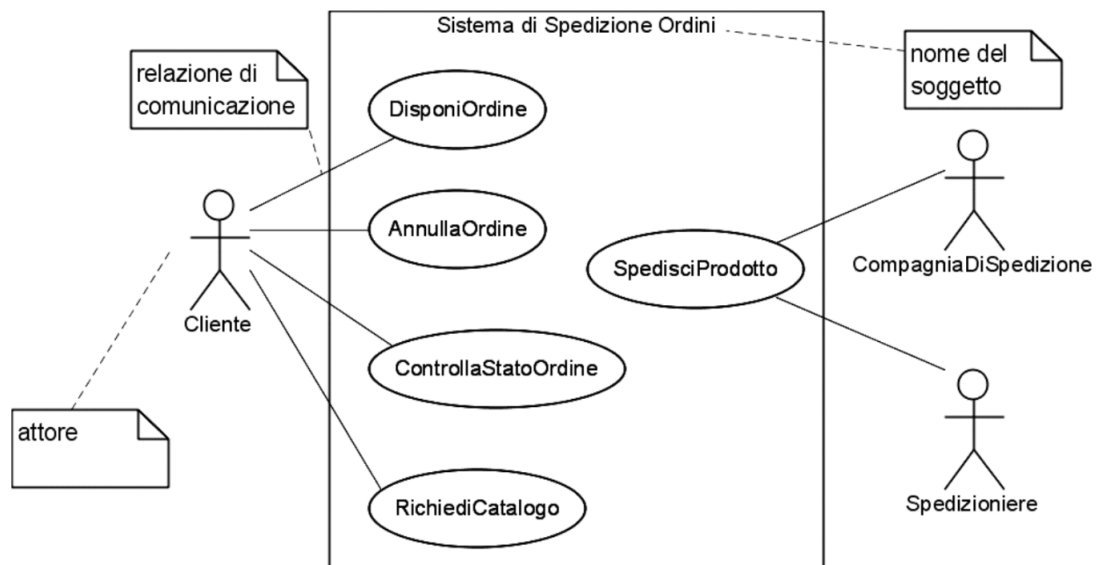
- Trovare gli *attori* coinvolti nell'uso del sistema, cioè il *ruolo* che le entità esterne assumono quando interagiscono *direttamente* col sistema.

Notiamo che nulla vieta che la stessa entità (fisica o astratta) assuma diversi ruoli nell'interazione col sistema. Ad esempio, possiamo pensare a un utente che si comporta sia da utilizzatore che da amministratore di un certo sistema.

In UML gli **attori** sono esterni ai *soggetti*. Potrebbe comunque essere che un sistema detiene una rappresentazione interna dell'attore (ad esempio una classe o un record di DB che mantiene i dati dell'utente);

- Trovare i **casi d'uso** del sistema, cioè il tipo di operazioni che il sistema dovrà compiere per conto degli utenti all'interno del suo dominio. A un caso d'uso è associato un *flusso* d'utilizzo del sistema da parte dell'utente. Flussi che divergono dal flusso di default vanno categorizzati e sono detti *flussi alternativi*.

Vediamo come la modellizzazione dei casi d'uso ci dà una visione d'insieme di come il sistema interagisce, nel suo dominio (*soggetto*) con le parti coinvolte (*attori*):



1.3.4 Flussi di casi d'uso

Un caso d'uso è quindi modellizzato attraverso una struttura tabulare che rispecchia la seguente:

Nome caso d'uso
Indice
Descrizione
Attore primario
Attori secondari
Precondizioni
Flusso principale:
1. Azione 1;
2. Azione 2;
3. ecc...
Postcondizioni
Flussi alternativi:
• Azione 2 fallita → Azione 3;
• ecc...

L'**indice** è solitamente progressivo (si possono adottare tassonomie, soprattutto nel caso di progetti complessi).

Un caso d'uso è sempre avviato da un singolo attore, l'attore **primario**. Questo non preclude il fatto che più attori possano avviare lo stesso flusso in momenti diversi. Inoltre, non preclude che altri attori vengano coinvolti: questi saranno gli attori **secondari**.

Le **precondizioni** sono le condizioni che devono verificarsi affinché il caso d'uso venga messo in moto, le **postcondizioni** le condizioni che questo lascerà una volta terminato.

Per definire i casi d'uso in UML usiamo ancora una sintassi molto semplice:

```
1 Il caso d'uso inizia quando un <attore> <funzione>
```

Il flusso di eventi è a questo punto una sequenza (nel caso più semplice):

```
1 <numero> Il <attore o altro> <azione>
```

Vediamo come è possibile prevedere dei **flussi alternativi** rispetto al **flusso principale** del caso d'uso. Questo permette una modellazione di comportamenti di errore, eccezioni, o semplici anomalie del flusso principale che si vogliono modellizzare. Per flussi più complicati ci è concesso usare altri costrutti più tipici della programmazione strutturata, cioè:

- Costrutti di ripetizione (for, while, ecc...);
- Costrutti condizionali (if, ecc...).

I flussi alternativi possono attivarsi in 3 modi differenti:

- Per scelta deliberata dell'attore principale;
- Attivato dopo un passo del flusso principale, in questo caso si specifica:

```
1 Il flusso alternativo comincia dopo il passo <numero> del flusso principale
```

- Attivato ad un passo qualsiasi del flusso principale, in questo caso si specifica:

```
1 Il flusso alternativo comincia in qualsiasi momento
```

Chiaramente, in ogni caso deve esserci una condizione che si verifica perché il flusso alternativo cominci.

1.3.5 Confronto fra requisiti e casi d'uso

Una volta terminata l'analisi dei requisiti e dei casi d'uso, si può procedere a stabilire le relazioni che collegano queste 2 categorie (una relazione molti a molti). Strumento utile in questo caso è la **matrice di tracciabilità**:

	Casi d'uso		
Requisiti	X		X
		X	
			X

1.3.6 Glossario di progetto

Il *glossario di progetto* è uno dei deliverable principali della fase di ingegnerizzazione dei requisiti. Questo fornisce un dizionario di termini chiave e definizioni usate nel dominio di applicazione, comprensibili a chiunque sia coinvolto nel progetto. Di fondamentale importanza è individuare i **sinonimi**, che potrebbero essere innumerevoli e non apparentemente equivalenti. Non meno importanti sono gli **omonimi**, cioè parole uguali usate con significati diversi.

Un esempio di glossario di progetto, riferito ad un sistema per la gestione dei prospetti di laurea, potrebbe essere il seguente:

Termine	Significato
Unità didattica (sinonimi: Segreteria)	La segreteria che interagisce col sistema, fornendo corso di laurea, data di laurea e elenco matricole e ricevendo in cambio i prospetti per la commissione e la possibilità di inviare i prospetti agli studenti.
Prospetti	Gli artefatti che il sistema genera, rappresentati da un frontespizio, una tabella di esami con informazioni associate, informazioni sulle medie e nel caso di prospetti destinati alla commissione, proposte di voto. Le proposte di voto vengono calcolate secondo formule specifiche ad ogni corso di laurea.
Commissione	Coloro che ricevono il prospetto dall'unità didattica, scaricato direttamente dalla pagina del sistema. Il loro prospetto è comprensivo di proposte di voto.