Appunti Ingegneria del Software

Luca Seggiani
2025

1 Lezione del 22-09-25

1.1 Introduzione

Il corso di ingegneria del software si pone di fornire gli strumenti per partire dalle conoscenze tecniche relative allo sviluppo di software e arrivare allo sviluppo e alla manutenzione di **prodotti** veri e propri.

Cercheremo quindi di gestire **sistemi** *"ingegneristici"*, cioè sistemi su larga scala e stratificati, dove non possiamo usare le stesse tecniche che funzionavano su semplici prototipi per gestire ogni singola parte a sé stante. Di fondamentale importanza in questo aspetto è iniziare a vedere il software come un insieme *"vivo"* e **modulare** di più *componenti* fra di loro interconnessi, fra cui ognuno potrebbe, nel suo ciclo di vita, *evolversi* in maniera differente.

Approfondiremo quindi più **fasi** di sviluppo di un prodotto software, e le regole relative allo sviluppo di software che sia facile da documentare, aggiornare e mantenere.

In questo sfrutteremo in primis le tecniche della **programmazione ad oggetti** (**OOP**, *Object Oriented Programming*), cercando però di motivare nel contesto di un sistema più complesso l'uso di strumenti come *ereditarietà*, *polimorfismo*, ecc...

1.1.1 Interazione col cliente

Prerogativa dello sviluppo di prodotti software al giorno d'oggi è la sintonia col *cliente* che richiede il prodotto stesso, e quindi che definisce quelle che sono le **specifiche** di progetto. Diversi framework sono stati sviluppati per favorire la comunicazione fra cliente e sviluppatori (*Agile*, ecc...), e la figura del **software analyst**, cioè *analista software*, è diventata più centrale alla professione di ingegnere software di quanto lo è quella del semplice sviluppatore.

Questa situazione è dovuta al fatto che le specifiche di progetto non sono, nelle prime fasi di progettazione, chiare nemmeno al cliente. Il compito dell'ingegnere software è quindi in primo luogo **documentale**, cioè mirato a riassumere le volontà del cliente (espresse in linguaggio naturale) in una serie di specifiche e quindi una struttura di progetto (espressa in un linguaggio appositamente sviluppato per la modellazione, cioè l'*UML*).

2 Lezione del 26-09-25

2.1 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.

2.2 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 partira 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).

2.2.1 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 dove si delina 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.

2.3 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:

```
<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 appropiati 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.

2.3.1 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 tmepo);
- **Want to have**: requisiti che non verranno realizzati adesso, ma al massimo in successive release.

2.3.2 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*.

2.3.3 Modellizzazione casi d'uso

La modellizzazione dei casi d'uso fa parte dell'ingegnerizzazione dei requisiti e procede 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.

  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*.

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: • Azione 1; • Azione 2; • ecc...
Postcondizioni
Flussi alternativi: • Azione 2 fallita → Azione 3; • ecc...

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**.

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

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

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

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

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:

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

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

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

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

2.3.4 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**:

Requisiti \ Casi d'uso			
	X		X
		X	
			X

2.3.5 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.

3 Lezione del 03-10-25