

# Sviluppo Applicazioni Software

Luca Barra

Anno accademico 2023/2024



# INDICE

<b>CAPITOLO 1</b>	<b>PROCESSI PER LO SVILUPPO SOFTWARE</b>	<b>PAGINA 1</b>
1.1	Introduzione Specifica dei requisiti — 2 • Sviluppo del software — 2 • Convalida del software — 2 • Evoluzione del software — 3	1
1.2	Modelli di processo software Modello a cascata — 3 • Modello incrementale — 4 • Integrazione e configurazione — 5 • Sviluppo incrementale, iterativo ed evolutivo — 5	3
1.3	Sviluppo agile I principi dello sviluppo agile — 6 • eXtreme Programming (XP) — 6 • Scrum — 7	5
<b>CAPITOLO 2</b>	<b>UNIFIED PROCESS (UP)</b>	<b>PAGINA 8</b>
2.1	OOA e OOD UML — 10	8
2.2	Unified Process Le discipline — 11 • Che cosa sono i requisiti? — 12	10
2.3	Ideazione	13
2.4	Casi d'Uso	13
2.5	Elaborazione	13
2.6	Modello di Dominio	13
2.7	Diagrammi di sequenza del sistema (DSS)	13
2.8	Contratti	13
<b>CAPITOLO 3</b>	<b>ARCHITETTURA DEL SOFTWARE</b>	<b>PAGINA 14</b>
<b>CAPITOLO 4</b>	<b>DIAGRAMMI DI INTERAZIONE E DI CLASSE UML</b>	<b>PAGINA 15</b>
<b>CAPITOLO 5</b>	<b>PATTERN GRASP</b>	<b>PAGINA 16</b>
<b>CAPITOLO 6</b>	<b>PATTERN GOF</b>	<b>PAGINA 17</b>
<b>CAPITOLO 7</b>	<b>DAL PROGETTO AL CODICE</b>	<b>PAGINA 18</b>



# Capitolo 1

## Processi per lo sviluppo software

### 1.1 Introduzione

In questa sezione verranno mostrati, anche in chiave storica, i principali processi per lo *sviluppo software*, *modelli di processi software*, sviluppo *iterativo* ed evolutivo, sviluppo *agile*.

#### Definizione 1.1.1: Software di qualità

- Non è un semplice programma o gruppo di programmi;
- Include *documentazione*, *test*, *manutenzione*, *aggiornamenti*;

#### Corollario 1.1.1 Caratteristiche essenziali

- ⇒ *Mantenibilità*: il software deve evolversi in base alle necessità dei clienti<sup>a</sup>;
- ⇒ *Fidatezza*: il software non dovrebbe causare danni fisici o economici;
- ⇒ *Efficienza*: il software deve fare un uso efficiente delle risorse;
- ⇒ *Accettabilità*: il software deve essere comprensibile, usabile e compatibile con altri sistemi.

<sup>a</sup>Da questo si hanno i maggiori introiti.

#### Note:-

A volte può convenire vendere il software "sottoprezzo" per poi guadagnare con la manutenzione.

#### Question 1

Cosa descrive un processo software?

**Risposta:** descrive *chi* fa *che cosa*, *come* e *quando* per raggiungere un *obiettivo*.

#### Definizione 1.1.2: Un processo per lo sviluppo software

Un processo software descrive un approccio *disciplinato* alla *costruzione*, al *rilascio* ed eventualmente alla *manutenzione* del software.

Si possono distinguere quattro attività di processo comuni:

- ⇒ *Specifiche del software*: clienti e sviluppatori definiscono le funzionalità del software (e i relativi vincoli);
- ⇒ *Sviluppo del software*: il software viene progettato e sviluppato;

- ⇒ *Convalida del software*: il software viene convalidato per garantire che soddisfi le specifiche del cliente;
- ⇒ *Evoluzione del software*: il software viene modificato per riflettere i cambiamenti nei requisiti del cliente e del mercato.

### 1.1.1 Specifica dei requisiti

Anche detta "*ingegneria dei requisiti*", è l'attività per capire e definire quali sono i requisiti richiesti dal sistema e identificare i vincoli all'operabilità e allo sviluppo del sistema.

**Le fasi principali di questa attività sono:**

- ⇒ *Deduzione e analisi dei requisiti*: osservazione di sistemi esistenti, discussioni con possibili utenti, analisi, etc.
- ⇒ *Specifica dei requisiti*: si traducono le informazioni raccolte in un *documento*;
- ⇒ *Convalida dei requisiti*: si controlla che i requisiti siano realistici, coerenti e completi.

### 1.1.2 Sviluppo del software

Anche detta "*progettazione e implementazione del software*", è l'attività di conversione delle specifiche del software in un sistema da consegnare al cliente. Nelle *metodologie agili* la progettazione e l'implementazione sono spesso *integrate* e, tipicamente, non producono documenti formali.

**Le fasi principali di questa attività sono:**

- ⇒ *Progettazione dell'architettura*: identifica la struttura complessiva del sistema, dei componenti, delle loro relazioni e della loro distribuzione;
- ⇒ *Progettazione del database*: si progetta la rappresentazione delle strutture dati che verranno utilizzate e la loro rappresentazione in un database<sup>1</sup>;
- ⇒ *Progettazione dell'interfaccia*: definisce l'interfaccia utente e le modalità di interazione con il sistema<sup>2</sup>;
- ⇒ *Progettazione e scelta dei componenti*: si ricercano i componenti riutilizzabili o vengono progettati nuovi componenti.

#### Note:-

La scelta dei componenti è particolarmente facile nel caso di linguaggi object-oriented.

### 1.1.3 Convalida del software

L'attività di verifica e convalida serve a dimostrare che un sistema sia *conforme* alle specifiche e che *soddisfi* le esigenze del cliente. La convalida richiede anche attività di *controllo*, *ispezione* e *revisione* a ogni stadio del processo di sviluppo. In alcune metodologie agili si scrivono i test prima di scrivere il codice (eXtreme Programming).

#### Note:-

In questo corso ci si concentrerà sul processo di testing. Per un modo formale di verificare la correttezza di un sistema si può fare riferimento al corso "Metodi formali dell'informatica".

**I test possono essere:**

- ⇒ *Test di unità (o dei componenti)*: i componenti vengono testati singolarmente<sup>3</sup>;
- ⇒ *Test del sistema*: si testa il sistema nel suo complesso;
- ⇒ *Test del cliente*: il sistema viene testato dal cliente con i propri dati.

<sup>1</sup>Non verrà trattata in questo corso. È stata parzialmente trattata nel corso "Basi di dati".

<sup>2</sup>Non verrà trattato lo sviluppo di un'interfaccia. È stato parzialmente trattato in "Programmazione III".

<sup>3</sup>Visti nel corso "Algoritmi e strutture dati".

### 1.1.4 Evoluzione del software

Anche detto "*manutenzione del software*", è l'attività di modifica durante o dopo lo sviluppo di un sistema software. La distinzione (storica) tra sviluppo e manutenzione è sempre più irrilevante. L'ingegneria del software è un unico processo evolutivo.

**Note:-**

Può capitare che si debba far fronte a cambiamenti improvvisi per esigenze di mercato o per incomprensioni con il cliente.

Bisogna *ridurre* i costi di rilavorazione:

- ⇒ *Anticipazione dei cambiamenti*: si possono prevedere o anticipare eventuali cambiamenti prima di una richiesta di rilavorazione;
- ⇒ *Tolleranza ai cambiamenti*: si progetta il sistema in modo da rendere facili eventuali cambiamenti.

Ci sono due metodi per far fronte ai cambiamenti:

- ⇒ *Prototipazione del sistema*: il sistema viene sviluppato rapidamente per verificare i requisiti del cliente. Ciò consente eventuali modifiche prima di sviluppare il sistema completo;
- ⇒ *Consegna incrementale*: vengono consegnati al cliente parti del sistema in modo incrementale in modo che il cliente possa provarlo e commentarlo.

**Note:-**

Il refactoring è un importante meccanismo per supportare la tolleranza ai cambiamenti

## 1.2 Modelli di processo software

Esistono veri modelli di processo software: *cascata*, *Unified Process*, *Scrum*, *XP*, *RUP*, *RAD*, *Spirale*, etc. Le quattro attività fondamentali sono organizzate in modo diverso in ciascun modello: in sequenza nel modello a cascata e intrecciate negli altri (modelli incrementali). Un ulteriore modello è il modello a integrazione e configurazione che però è poco trattato a livello ingegneristico.

### Definizione 1.2.1: Paradigma di processo

Il modello di processo software è una rappresentazione semplificata di un processo software. Sono strutture di processo da *estendere* e *adattare* per soddisfare le esigenze specifiche di un progetto.

**Note:-**

Non esiste un modello di processo software "universale", ma la scelta del modello dipende dai requisiti del cliente:

- ⇒ i software a sicurezza critica richiedono un modello a cascata per via delle analisi e della documentazione;
- ⇒ i software per il mercato richiedono un modello incrementale;
- ⇒ i sistemi aziendali richiedono un modello a configurazione e integrazione.

Inoltre, in grandi sistemi, si possono combinare più modelli.

### 1.2.1 Modello a cascata

#### Definizione 1.2.2: Modello a cascata

Il modello a cascata è un modello di processo software in cui le fasi di sviluppo sono viste come *fasi distinte* e *non sovrapposte*. Questo modello era l'unico modello utilizzato fino agli anni '80.

**Note:-**

Si contrappone ai modelli incrementali in cui le fasi di sviluppo sono sovrapposte e iterate.

**Corollario 1.2.1** Fasi del modello a cascata

- ⇒ All'inizio si definiscono i requisiti;
- ⇒ All'inizio si definisce un piano temporale;
- ⇒ Si progetta e modella il sistema;
- ⇒ Si crea un progetto completo del software;
- ⇒ Si inizia la programmazione del sistema;
- ⇒ Si testa il sistema, si rilascia e si prosegue con la manutenzione.

**Il modello a cascata:**

- ⇒ Non è adatto allo sviluppo in team;
- ⇒ Si dovevano definire spesso modelli matematici;
- ⇒ Costava molto in termini di tempo e denaro.

**1.2.2 Modello incrementale**

**Definizione 1.2.3: Modello incrementale**

Il modello incrementale è un modello di processo software in cui il sistema viene sviluppato in *incrementi* (o *iterazioni*). Si effettuano *feedback veloci* e *rilasci*.

**Note:-**

Negli anni '80 e '90 molte persone si avvicinano al mondo della progettazione e nasce la necessità di sviluppare software in modo incrementale.

**Corollario 1.2.2** I casi d'uso

I casi d'uso sono il modo migliore per definire i requisiti: il cliente racconta una storia e il programmatore la traduce in un caso d'uso.

**Lo sviluppo incrementale:**

- ⇒ È un approccio *plan-driven*, *agile* o una combinazione di questi approcci;
- ⇒ Se *plan-driven*, si pianificano in anticipo gli incrementi;
- ⇒ Se *agile*, si identificano gli incrementi iniziali ma si dà priorità al rilascio di incrementi che soddisfano i requisiti più importanti;
- ⇒ Il costo di implementazione di modifiche è ridotto;
- ⇒ È più facile ottenere un feedback dal cliente;

**Note:-**

Tuttavia si devono avere consegne regolari e frequenti, la struttura dei sistemi tende a degradarsi e richiede pianificazione in anticipo per grandi team.



### 1.2.3 Integrazione e configurazione

#### Definizione 1.2.4: Riutilizzo del software

- ⇒ Dagli anni 2000 si sono diffusi software che riutilizzano software già esistente;
- ⇒ Collezioni di oggetti che sono sviluppati come un componente o un pacchetto da integrare tramite framework;
- ⇒ Servizi web che possono essere integrati in un sistema.

Le fasi principali sono:

- ⇒ *Specifica dei requisiti*;
- ⇒ *Ricerca e valutazione del software*: se esiste un software che soddisfa i requisiti;
- ⇒ *Perfezionamento dei requisiti*: utilizzando le informazioni trovate nella ricerca;
- ⇒ *Configurazione del sistema di applicazioni*;
- ⇒ *Adattamento e integrazione*: si integra il sistema con i componenti riutilizzabili.

#### Note:-

Questo approccio riduce la quantità di software da sviluppare, riducendo i costi e i rischi. Però bisogna scendere a compromessi con i requisiti e si perde il controllo sull'evoluzione del sistema.

### 1.2.4 Sviluppo incrementale, iterativo ed evolutivo

Questo modello è:

- **Incrementale**: si incrementa il codice man mano che si sviluppa;
- **Iterativo**: si sviluppa il software in cicli (iterazioni);
- **Evolutivo**: si sviluppa il software in modo che possa evolvere a ogni iterazione richiedendo un feedback.

#### Definizione 1.2.5: Approccio iterativo

Nell'approccio iterativo:

- ⇒ lo sviluppo è organizzato in mini-progetti brevi (le iterazioni);
- ⇒ il risultato di ogni iterazione è un sistema parzialmente funzionante (testato e integrato);
- ⇒ ogni iterazione dura poche settimane<sup>a</sup> e comprende le proprie attività di analisi, sviluppo, etc.;
- ⇒ si ottiene un feedback a ogni iterazione.

<sup>a</sup>Un'iterazione di lunghezza fissata è detta *timeboxed*.

#### Note:-

Git supporta lo sviluppo incrementale, iterativo ed evolutivo.

## 1.3 Sviluppo agile

#### Definizione 1.3.1: Sviluppo agile

Lo sviluppo *agile* è un insieme di metodi di sviluppo software.

### Contesto:

- ⇒ Il software è parte essenziale delle operazioni aziendali;
- ⇒ La rapidità della consegna è un fattore critico;
- ⇒ Spesso non si possono ottenere requisiti stabili;
- ⇒ I requisiti diventano chiari solo dopo che il sistema è stato consegnato e utilizzato;
- ⇒ In successive iterazioni si possono ottenere requisiti più chiari.

### 1.3.1 I principi dello sviluppo agile

#### Definizione 1.3.2: Agile Modelling

Lo scopo della modellazione (UML) è principalmente quello di *comprendere* e di agevolare la *comunicazione*, non di documentare.

- ⇒ Adottare un metodo agile non significa evitare del tutto la modellazione;
- ⇒ Non si deve applicare UML per eseguire per intero o per la maggior parte la progettazione software;
- ⇒ Va utilizzato l'approccio più semplice e che comporta il minor dispendio di energie. Esempio: abbozzo di UML su una lavagna;
- ⇒ La modellazione non va fatta da soli ma in coppie o in gruppo;
- ⇒ Solo il codice verificato dimostra il vero progetto, i diagrammi precedenti sono suggerimenti incompleti (usa e getta);
- ⇒ La modellazione per OO dovrebbe essere eseguita dagli stessi sviluppatori che andranno effettivamente a scrivere il codice.

### Pratiche innovative:

- ⇒ *Storie utente*: scenari d'uso in cui potrebbe trovarsi un utente. Il cliente lavora a stretto contatto con il team di sviluppo e discute di possibili scenari;
- ⇒ *Refactoring*: il codice va costantemente rifattorizzato per proteggerlo dal deterioramento causato dallo sviluppo incrementale;
- ⇒ *Sviluppo con test iniziali*: lo sviluppo non può procedere finché tutti i test non sono stati superati;
- ⇒ *Programmazione a coppie*: i programmatori lavorano a coppie nella stessa postazione per sviluppare il software.

### 1.3.2 eXtreme Programming (XP)

#### Definizione 1.3.3: eXtreme Programming

eXtreme Programming (XP) è un metodo di sviluppo software che si basa su valori e principi di base:

- ⇒ sviluppo incrementale attraverso piccole e frequenti release;
- ⇒ il cliente è parte attiva dello sviluppo;
- ⇒ il progetto è supportato da test, refactoring e integrazione continua;
- ⇒ si punta a mantenere la semplicità.

### 1.3.3 Scrum

#### Definizione 1.3.4: Scrum

Scrum offre un framework per organizzare progetti agili e fornire una visibilità esterna su ciò che sta accadendo, ossia si occupa dell'organizzazione del lavoro e della gestione dei progetti.

Scrum è un approccio iterativo e incrementale in cui ciascuna iterazione ha una durata fissata denominata Sprint (non si hanno estensioni).

Sono presenti tre ruoli:

- ⇒ *Product Owner*: rappresenta il cliente, definisce i requisiti e specifica le priorità attraverso il *Product Backlog*<sup>4</sup>;
- ⇒ *Development Team*: le persone che sviluppano il software;
- ⇒ *Scrum Master*: garantisce che il team segua le regole di Scrum.

Gestione agile della progettazione:

- ⇒ Il Development Team seleziona dal Product Backlog un insieme di voci da sviluppare durante quell'iterazione (*Sprint Goal*), compila lo *Sprint Backlog* (ossia i compiti dettagliati per raggiungere il goal);
- ⇒ Il risultato di ciascuno Sprint è un prodotto software funzionante chiamato "incremento di prodotto potenzialmente rilasciabile" (integrato, verificato e documentato);
- ⇒ Nello *Sprint Review* il Product Owner e il Development Team presentano le parti coinvolte dall'incremento, ne fanno la dimostrazione, ottengono un feedback e decidono cosa fare nello Sprint successivo;
- ⇒ Si dà enfasi all'adozione di Team auto-organizzati e auto-gestiti.

---

<sup>4</sup>Un elenco di voci, funzionalità e requisiti.

# Capitolo 2

## Unified Process (UP)

### 2.1 OOA e OOD

#### Definizione 2.1.1: OOA/D

- **OOA** (Object Oriented Analysis): studio dei requisiti e delle specifiche del sistema;
- **OOD** (Object Oriented Design): progettazione del sistema.

Per studiare OOA/D si utilizza Unified Process, un processo di sviluppo software orientato agli oggetti.

#### Note:-

UP può essere applicato usando un approccio agile come Scrum o XP.

#### Corollario 2.1.1 UML

UP utilizza UML come linguaggio di modellazione. UML è un linguaggio di modellazione grafico e testuale per la specifica, la costruzione e la documentazione di sistemi software orientati agli oggetti.

**IMPORTANTE:** UML non è nato per descrivere software, ma per descrivere concetti<sup>a</sup>.

<sup>a</sup>Simile a ER, visto nel corso "Basi di dati".

**OOD è guidata dalle responsabilità (si vedano i pattern GRASP):**

- ⇒ Quali sono gli oggetti? Quali sono le classi?
- ⇒ Cosa deve conoscere un oggetto? Cosa deve saper fare?
- ⇒ Come collaborano gli oggetti?

#### Definizione 2.1.2: Pattern

I pattern sono euristiche, best practice, che aiutano a codificare principi di soluzioni.

**OOD è correlata all'analisi dei requisiti:**

- ⇒ *Casi d'uso*;
- ⇒ *Storie utente*.

### Esempio 2.1.1 (Gioca una *partita a dadi*)

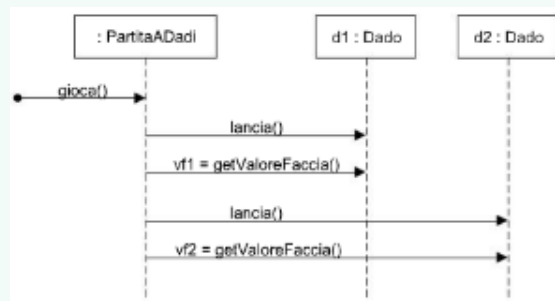
**Definizione dei casi d'uso: storie scritte.**

Il *Giocatore* chiede di *lanciare* i *dadi*. Il Sistema presenta il *risultato*: se *il valore totale* delle facce dei dadi è sette, il giocatore ha vinto; altrimenti ha perso.

**Definizione di un modello di dominio: *i concetti o gli oggetti significativi*.**



**Assegnare responsabilità agli oggetti e disegnare diagrammi di interazione: *responsabilità e collaborazioni*.**



**Definizione dei diagrammi delle classi di progetto.**



L'analisi dei requisiti e l'OOA/D vanno svolte nel contesto di un processo di sviluppo:

- ⇒ Sviluppo iterativo;
- ⇒ Approccio agile;
- ⇒ Unified Process (UP).

**Note:-**

ER e UML non sono pienamente adatti a possibili incrementi.

### 2.1.1 UML

#### Definizione 2.1.3: UML

UML è un linguaggio *visuale* per la specifica, la costruzione e la documentazione degli elaborati di un sistema software.

UML è uno standard per la notazione di diagrammi per disegnare o rappresentare figure relative al software (specialmente OO).

UML è un *abbozzo* o un *progetto* per aiutare la comprensione nei team di sviluppo. Il termine abbozzo indica che può essere soggetto a correzione, ma se non ci sono feedback a tal proposito deve essere trattato come un dizionario.

#### Uso di UML:

- ⇒ Punto di vista *concettuale*: modello di dominio, per visualizzare concetti del mondo reale;
- ⇒ Punto di vista *software*: diagramma delle classi di progetto, utilizzata per visualizzare elementi software.

#### Brevi note storiche:

- ⇒ Anni '60 e '70: nascita dei linguaggi OO (Simula e Smalltalk);
- ⇒ 1988: Bertrand Meyer, "Object-Oriented Software";
- ⇒ 1991: Jim Rumbaugh, "Object-Oriented Modelling and Design" (OOA/D);
- ⇒ 1991, Grady Booch, "Object-Oriented Software Engineering" (OOA/D e Casi d'Uso);
- ⇒ 1994, Rumbaugh e Booch fanno le prime proposte di UML;
- ⇒ Rational Corporation fondata dai "tre amigos" (Jacobson, Booch e Rumbaugh);
- ⇒ 1997 UML 1;
- ⇒ 2004 UML 2 (usato attualmente).

## 2.2 Unified Process

#### Definizione 2.2.1: Unified Process

Unified Process è un processo iterativo ed evolutivo (incrementale) per lo sviluppo del software per la costruzione di sistemi orientati agli oggetti. Le iterazioni iniziali sono guidate dal *rischio*, dal *cliente* e dall'*architettura*.

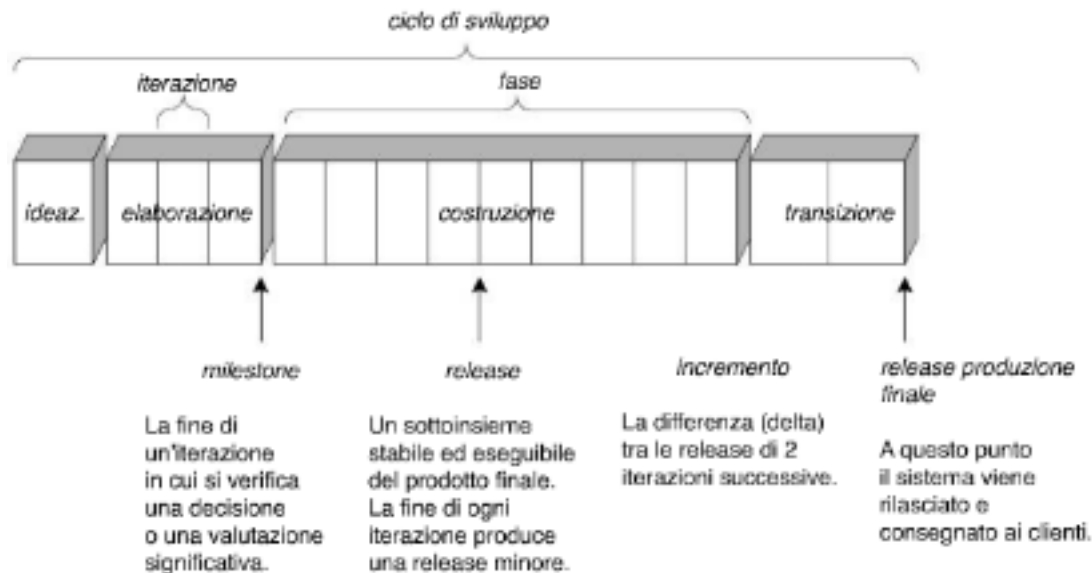
#### Question 2

Cosa c'è in UP?

- ⇒ Un'organizzazione del piano di progetto per fasi sequenziali;
- ⇒ Indicazioni sulle attività da svolgere nell'ambito di discipline e sulle loro inter-relazioni;
- ⇒ Un insieme di ruoli predefiniti;
- ⇒ Un insieme di artefatti da produrre.

Un progetto UP è organizzato in 4 fasi:

- ⇒ **Ideazione** (inception): visione approssimativa, studio economico, portata, stime approssimative di costi e tempi. **Milestone:** *Obiettivi*;
- ⇒ **Elaborazione** (elaboration): visione raffinata, implementazione iterativa del nucleo dell'architettura, risoluzione dei rischi maggiori, identificazione della maggior parte dei requisiti e della portata, stime più realistiche sulle loro inter-relazioni. **Milestone:** *Architetturale*;
- ⇒ **Costruzione** (construction): implementazione iterativa degli elementi rimanenti, più facili e a rischio minore, preparazione al rilascio. **Milestone:** *Capacità operativa*;
- ⇒ **Transizione** (transition): beta test, rilascio. **Milestone:** *Rilascio prodotto*.



**Note:-**

- ⇒ L'Ideazione non è una fase di requisiti, ma di fattibilità;
- ⇒ L'Elaborazione non è una fase di requisiti o di progettazione, ma una fase in cui si implementa in modo iterativo l'architettura del sistema e vengono ridotti i rischi maggiori.

## 2.2.1 Le discipline

### Definizione 2.2.2: Discipline

Una disciplina è un insieme di attività e dei relativi *elaborati* in una determinata area, come le attività relative all'analisi dei requisiti.

#### Corollario 2.2.1 Elaborato

Un elaborato (artefatto o work product) è il termine generico che indica un qualsiasi prodotto di lavoro: codice, schemi di basi di dati, documenti di testo, diagrammi, modelli, etc.

### Discipline ingegneristiche di UP:

- ⇒ **Modellazione del business**: attività che modellano il dominio del problema e il suo ambito;
- ⇒ **Requisiti**: attività di raccolta dei requisiti;
- ⇒ **Progettazione** (analysis and design): attività di analisi dei requisiti e progetto architetturale;

- ⇒ *Implementazione*: attività di progetto dettagliato e codifica del sistema, test sui componenti;
- ⇒ *Test*: attività di controllo di qualità, test di integrazione e di sistema;
- ⇒ *Rilascio*: attività di consegna e messa in opera.

#### Discipline di supporto di UP:

- ⇒ *Gestione delle configurazioni e del cambiamento*: attività di manutenzione durante il progetto;
- ⇒ *Gestione progetto*: attività di pianificazione e governo del progetto;
- ⇒ *Infrastruttura* (environment): attività che supportano il team di progetto, riguardo ai processi e strumenti utilizzati.

#### Note:-

Nonostante le fasi siano *sequenziali*, le discipline non lo sono (perchè si eseguono in ogni iterazione). Il numero di iterazioni dipende dal Project Manager.

#### Uso di UML in UP:

- ⇒ UP usa solo UML come linguaggio di modellazione;
- ⇒ I diagrammi UML si usano con variabilità, bisogna *personalizzare* UP;
- ⇒ I diagrammi si usano in UP seguendo le iterazioni e gli incrementi;
- ⇒ UP dice *quando* usare un diagramma;
- ⇒ In UP quasi tutto è *opzionale* eccetto che lo sviluppo iterativo e guidato dal rischio, la verifica continua della qualità e il codice;
- ⇒ La scelta delle pratiche e degli artefatti UP si riassume in un documento (*scenario di sviluppo*).

### 2.2.2 Che cosa sono i requisiti?

#### Definizione 2.2.3: Requisito

Un requisito è una *capacità* o una condizione a cui il sistema deve essere *conforme*.

#### Corollario 2.2.2 Sorgenti dei requisiti

I requisiti derivano da richieste degli utenti del sistema per risolvere dei problemi e raggiungere degli obiettivi. Possono essere:

- ⇒ *Requisiti funzionali*: descrivono il comportamento del sistema in termini di funzionalità offerte;
- ⇒ *Requisiti non funzionali*: le proprietà del sistema nel suo complesso (sicurezza, prestazioni, etc.).

#### Note:-

In UP bisogna gestire i requisiti: si utilizza un approccio sistematico per trovare, documentare, organizzare e tracciare i requisiti che cambiano di un sistema. Si inizia a programmare quando sono stati specificati il 10% o il 20% dei requisiti significativi.

#### Acquisizione sistematica dei requisiti:

- ⇒ Scrivere i Casi d'Uso con i clienti;
- ⇒ Workshop dei requisiti con sviluppatori e clienti;
- ⇒ Gruppi di lavoro con rappresentanti dei clienti;
- ⇒ Dimostrazione ai clienti dei risultati di ciascuna iterazione, per favorire un feedback.



### Modello FURPS+:

- ⇒ *Funzionali* (F): requisiti funzionali e di sicurezza;
- ⇒ *Usabilità* (U): facilità d'uso del sistema;
- ⇒ *Affidabilità* (R - Reliability): disponibilità del sistema, capacità di tollerare guasti o di essere ripristinato;
- ⇒ *Prestazioni* (P): tempi di risposta, throughput, capacità e uso delle risorse;
- ⇒ *Sostenibilità* (S): facilità di modifica per riparazioni e miglioramenti, adattabilità, manutenibilità, localizzazione, configurazione, compatibilità;
- ⇒ *+*: vincoli di progetto, interoperabilità, operazionali, fisici, legali, etc.

### Elaborati:

- ⇒ *Modello dei Casi d'Uso*: scenari tipi dell'utilizzo di un sistema;
- ⇒ *Specifiche supplementari*: ciò che non rientra nei Casi d'Uso, requisiti non funzionali o funzionali non esprimibili attraverso i Casi d'Uso;
- ⇒ *Glossario*: termini significativi, dizionario dei dati;
- ⇒ *Visione*: riassume i requisiti di alto livello, un documento sintetico per apprendere rapidamente le idee principali del progetto;
- ⇒ *Regole di Business*: regole di dominio, i requisiti o le politiche che trascendono un unico progetto software e a cui un sistema deve conformarsi.

## 2.3 Ideazione

## 2.4 Casi d'Uso

## 2.5 Elaborazione

## 2.6 Modello di Dominio

## 2.7 Diagrammi di sequenza del sistema (DSS)

## 2.8 Contratti

## Capitolo 3

# Architettura del software

## Capitolo 4

# Diagrammi di interazione e di classe UML

## Capitolo 5

# Pattern GRASP

## Capitolo 6

# Pattern GOF

## Capitolo 7

# Dal progetto al codice