



# Analisi e Progettazione del Software

spitfire

A.A. 2023-2024

# Contents

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Introduzione all'ingegneria del software . . . . .	3
1.2	La crisi del software . . . . .	4
1.3	Analisi e progettazione . . . . .	5
1.3.1	Analisi e progettazione orientata agli oggetti . . . . .	5
1.4	Introduzione ai diagrammi e ai passi fondamentali dello sviluppo software	6
1.4.1	Definizione dei casi d'uso . . . . .	6
1.4.2	Definizione di un modello di dominio . . . . .	7
1.4.3	Definizione dei diagrammi di interazione . . . . .	7
1.4.4	Definizione dei diagrammi di classe di progetto . . . . .	7
1.5	UML . . . . .	8
1.5.1	UML e gli oggetti . . . . .	9
1.5.2	Tre modi di applicare UML . . . . .	9
1.5.3	Due punti di vista per applicare UML . . . . .	10
1.5.4	Significato di classe . . . . .	11
1.5.5	Vantaggi della modellazione visuale . . . . .	11
<b>2</b>	<b>Processi per lo sviluppo del software</b>	<b>11</b>

# 1 Introduzione

Che cos'è il **software**? Esso è un **programma per computer** unito alla **documentazione** ad esso associata, la quale specifica e comprende **requisiti, modelli di progetto, manuale utente,...**

I prodotti software possono essere:

- **Generici**: sviluppati per un ampio insieme di clienti (elaboratori di testo, database,...)
- **Personalizzati** (custom): sviluppati per un singolo cliente in base alle sue esigenze specifiche

Un nuovo prodotto software può essere **creato da zero, personalizzando software già esistenti o riusando parti di software già esistenti**. Le caratteristiche essenziali di un buon software sono:



MANTENIBILITÀ



FIDATEZZA



EFFICIENZA



ACCETTABILITÀ

## 1.1 Introduzione all'ingegneria del software

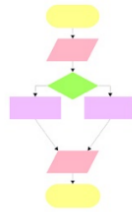
Che cos'è l'**ingegneria del software**? L'**ingegneria del software** è una disciplina ingegneristica che si occupa di tutti gli aspetti della produzione del software di buona qualità, dalle **prime fasi della specifica del sistema fino alla manutenzione del sistema** dopo la messa in uso. Vediamo cosa si intende per **disciplina ingegneristica** e "**Tutti gli aspetti della produzione del software**":

- **Disciplina ingegneristica**: Utilizzare metodi e teorie **appropriati** per risolvere i problemi tenendo conto dei vincoli **organizzativi e finanziari**
- **Tutti gli aspetti della produzione del software**: Non solo il **processo tecnico di sviluppo**. Anche la **gestione del progetto** e lo sviluppo di **strumenti**, metodi ecc... per supportare la produzione del software

La disciplina dell'ingegneria del software si occupa di:



Metodologie



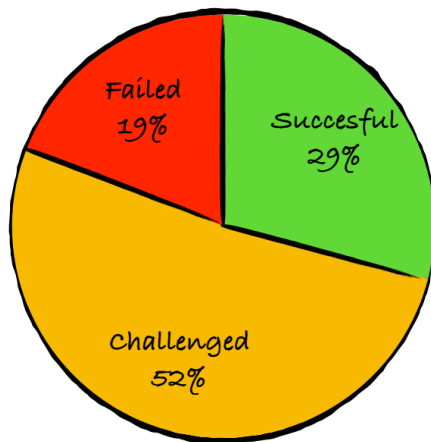
Tecniche



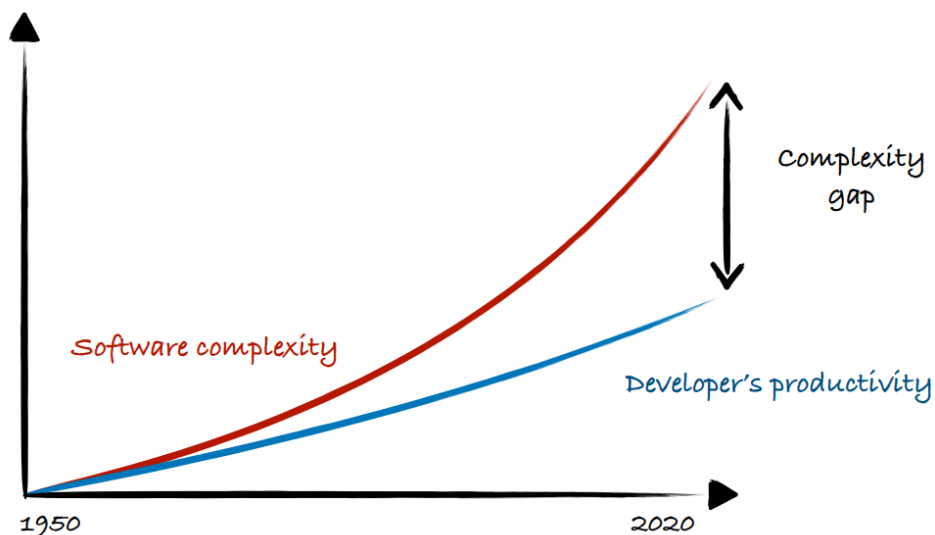
Strumenti

## 1.2 La crisi del software

Il termine **crisi del software** (o software crisis) è usato nell'ambito dell'ingegneria del software per descrivere l'impatto della **rapida crescita** della potenza degli elaboratori e la **complessità** dei problemi che dovevano esseri affrontati. Le parole chiavi della software crisis erano **complessità, attese e cambiamento**. Il concetto di software crisis emerse negli anni '60.



Standish Group 2015 Chaos Report



Le cause della crisi del software erano legate alla **complessità dei processi software** e alla **relativa immaturità dell'ingegneria del software**. Per superare la crisi infatti si dovettero introdurre:

- **Management**
- **Organizzazione**, attraverso **analisi e progettazione**
- **Teorie e tecniche** come la **programmazione strutturata e ad oggetti**
- **Strumenti**, come gli IDE
- **Metodologie**, tra cui il **modello a cascata** e il **modello agile**

### 1.3 Analisi e progettazione

Che cosa sono **analisi e progettazione**?

L'**analisi** enfatizza un'**investigazione del problema e dei requisiti** invece che una soluzione: per esempio, se si vuole realizzare un nuovo sistema di trading online, bisognerà capire **come questo sistema verrà utilizzato** e **quali sono le sue funzioni**. "Analisi" è un termine ampio con più accezioni, tra cui:

- **Analisi dei requisiti**, cioè un'investigazione dei requisiti del sistema
- **Analisi orientata agli oggetti**, cioè un'investigazione degli oggetti di dominio

La **progettazione** enfatizza una soluzione **concettuale** (software e hardware) che **soddisfa i requisiti**, anziché la relativa implementazione. Per esempio, la descrizione di uno schema di base di dati e di oggetti software. Nella progettazione vengono spesso **esclusi dettagli di basso livello o "ovvi"** (o almeno "ovvi" per coloro a cui è destinato il software).

Infine i progetti possono essere **implementati** e la loro implementazione (ovvero il codice) esprime il progetto realizzato vero e completo. Come nel caso dell'analisi, anche "progettazione" è un termine con più accezioni, tra cui:

- **Progettazione orientata agli oggetti**
- **Progettazione di basi di dati**

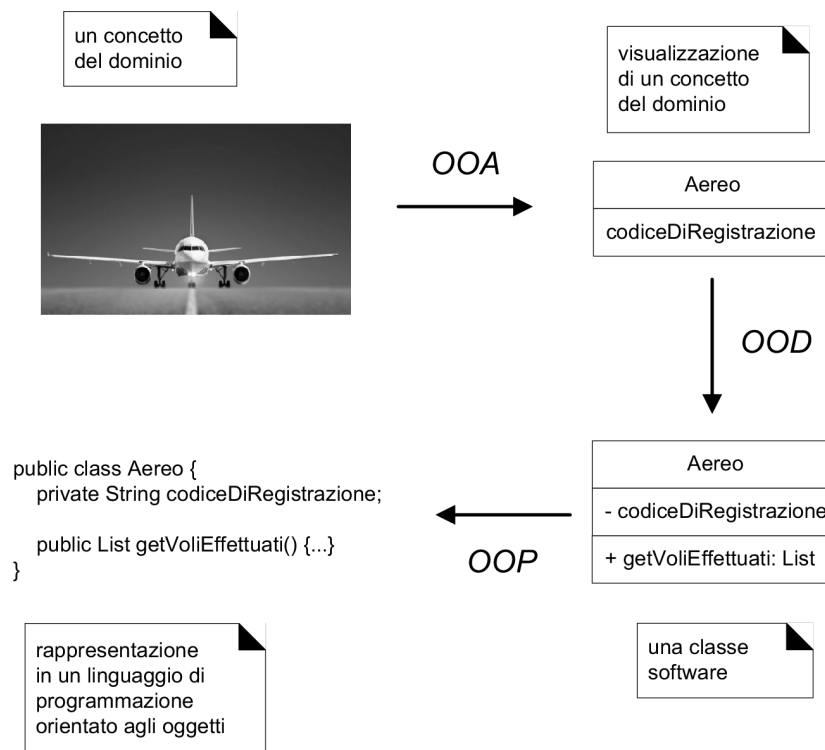
L'analisi e la progettazione possono essere riassunti con la seguente frase:

**Fare la cosa giusta**(analisi) e **fare la cosa bene**(progettazione)

#### 1.3.1 Analisi e progettazione orientata agli oggetti

Durante l'**analisi orientata agli oggetti** c'è un'enfasi sull'**identificazione** e la **descrizione degli oggetti**, o dei **concetti**, nel **dominio del problema**. Per esempio, nel caso di un sistema informatico per voli aerei, alcuni dei concetti possono essere *Aereo*, *Volo* e *Pilota*.

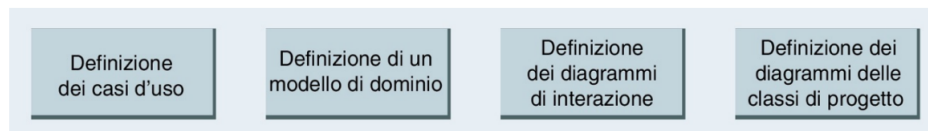
Durante la **progettazione orientata agli oggetti** (o più semplicemente **progettazione a oggetti**) l'enfasi è sulla **definizione di oggetti software** e sul **modo in cui questi collaborano per soddisfare i requisiti**. Per esempio un oggetto software *Aereo* può avere un attributo *codiceDiRegistrazione* e un metodo *getVoliEffettuati*.



Infine durante l'**implementazione** o la **programmazione orientata agli oggetti**, gli oggetti progettati vengono implementati, per esempio implementando la classe *Aereo* in un linguaggio ad oggetti. Dunque, analisi e progettazione **hanno obbiettivi diversi che vengono perseguiti in maniera diversa**. Tuttavia, come mostrato dall'esempio sopra, esse sono **attività fortemente sinergiche** che sono **correlate fra loro** e con le **altre attività dello sviluppo del software**.

## 1.4 Introduzione ai diagrammi e ai passi fondamentali dello sviluppo software

Vediamo una breve introduzione dei **vai diagrammi e dei passi fondamentali** legati allo sviluppo software.



### 1.4.1 Definizione dei casi d'uso

L'**analisi dei requisiti** può comprendere **storie o scenari** relativi al modo in cui l'applicazione può essere utilizzata dagli utenti; queste storie possono essere scritte come **casi d'uso**. I casi d'uso **non sono un elaborato ad oggetti** ma semplicemente delle storie scritte. Sono tuttavia uno strumento **diffuso nell'analisi dei requisiti**. Facciamo un'esempio:

**Gioca una partita a Dadi:** 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.

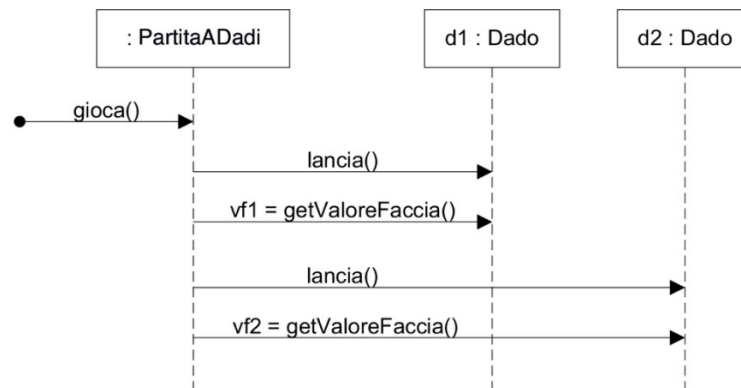
### 1.4.2 Definizione di un modello di dominio

L'analisi orientata agli oggetti è interessata alla **creazione di una descrizione del dominio da un punto di vista ad oggetti**. Vengono identificati i **concetti, gli attributi e le associazioni considerati significativi**. Il risultato può essere espresso come un **modello di dominio** che mostra i concetti o gli oggetti **significativi** del dominio. Esso è rappresentato nel seguente modo:



### 1.4.3 Definizione dei diagrammi di interazione

La **progettazione ad oggetti** è interessata alla **definizione di oggetti software, delle loro responsabilità e collaborazioni**. Una notazione comune per illustrare queste collaborazioni è un **diagramma di sequenza** (un tipo di diagramma UML). Esso mostra lo scambio di messaggi **tra oggetti software**, dunque l'invocazione di **metodi**. Esso è rappresentato nel seguente modo:

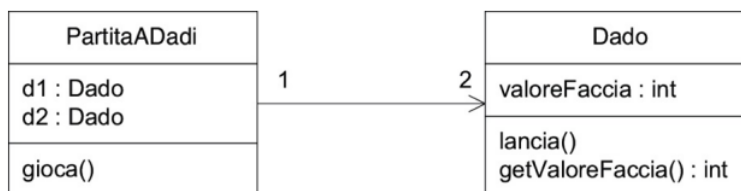


È interessante notare come la **progettazione degli oggetti software e dei programmi si può ispirare a un dominio del mondo reale**, tuttavia essa non è **nè un modello diretto nè una simulazione di questo dominio**. Quindi, per esempio, seppur nel mondo reale è il giocatore a lanciare il dado, nel progetto software è l'oggetto *PartitaADadi* che "lancia" i dadi.

### 1.4.4 Definizione dei diagrammi di classe di progetto

Accanto a una visione dinamica delle **collaborazioni tra oggetti**, mostrata dai diagrammi di interazione, è utile mostrare una **vista statica** delle definizioni di classi mediante un **diagramma delle classi di progetto**, che mostra le classi software con

i loro attributi e metodi. Il diagramma delle classi di progetto è rappresentato nella seguente maniera:

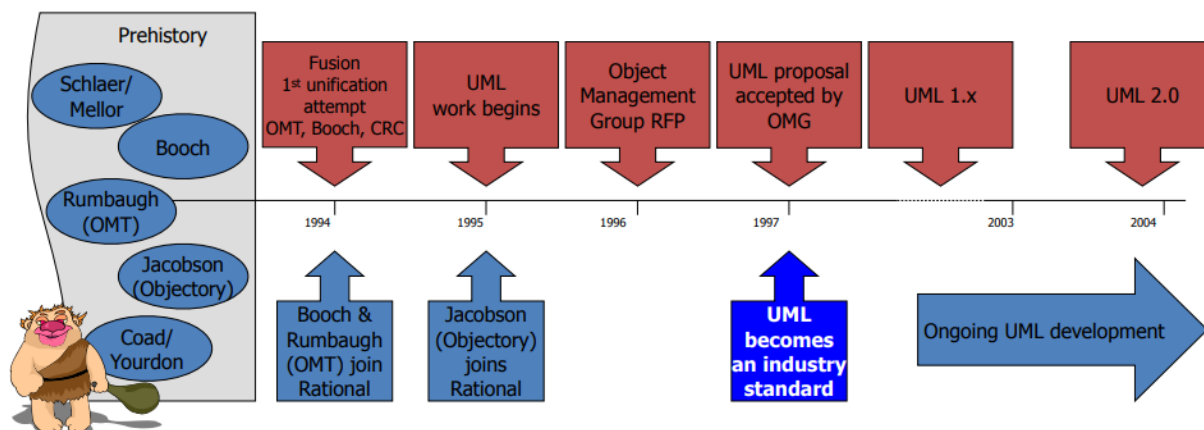


Diversamente dal modello di dominio, che illustra **classi del mondo reale**, questo diagramma mostra **classi software**. Si noti che, benché questo diagramma delle classi di progetto **non sia uguale al modello di dominio**, i nomi e il contenuto delle classi sono **simili**. In tal modo i **progetti e i linguaggi Object Oriented (OO)** sono in grado di **favorire un salto rappresentazionale basso** tra i componenti software e il nostro modello mentale di un dominio, **migliorando la comprensione**.

## 1.5 UML

**Unified Modelling Language**, abbreviato **UML**, è un **linguaggio visuale** per la **specifica, la costruzione e la documentazione degli elaborati** di un sistema software. UML rappresenta una **collezione di best practices di ingegneria**, dimostrate vincenti nella modellazione di sistemi vasti e complessi; inoltre esso **favorisce la divulgazione delle informazioni nella comunità dell'ingegneria del software** in quanto è *standard de facto*. Bisogna però tenere a mente che **UML non è una metodologia ma un linguaggio!**

Il termine *visuale* della definizione è un punto fondamentale. UML è uno standard de facto per la **notazione di diagrammi per disegnare o rappresentare figure** (con del testo) **relative al software**, e in particolare, al software OO. A un livello più profondo, di particolare interesse per i produttori di strumenti per **MDA** (Model Driven Architecture) alla base della notazione UML c'è il **meta-modello di UML** che descrive la **semantica** degli elementi di modellazione, tuttavia non è necessario che lo sviluppatore lo conosca. Presentiamo ora una breve storia di UML:



Il più significativo aggiornamento di UML è avvenuto nel **2003**:



- Maggiore **consistenza**
- Semantica definita in maniera **più chiara e dettagliata**
- **Nuovi diagrammi**
- Compatibilità con le precedenti versioni (1.x)

Altra parola importante è *unified*: UML vuole essere un **linguaggio unificante** sotto diversi aspetti:

- **Storico** (OMT, Booch, CRC, Objectory)
- **Ciclo di sviluppo** (sintassi visuali per tutte le fasi)
- **Domini applicativi** (dai sistemi embedded ai sistemi gestionali)
- **Linguaggi e piattaforme di sviluppo** (.Net, Java, C#,...)
- **Processi di sviluppo** (UP, BPM, ...)

### 1.5.1 UML e gli oggetti

UML modella i sistemi come **una serie di oggetti che collaborano fra loro**. Si hanno quindi due strutture:

- **Struttura statica:**
  - **Quali** tipi di oggetti sono necessari
  - **Come** sono correlati
- **Struttura dinamica:**
  - **Ciclo di vita** di questi oggetti
  - **Come collaborano** per fornire le funzionalità richieste

### 1.5.2 Tre modi di applicare UML

Fowler [Fowler03] descrive tre modi per applicare UML:

- **UML come abbozzo:** Diagrammi **informali e incompleti** (spesso abbozzati a mano su una lavagna bianca), che vengono creati per **esplorare parti difficili dello spazio del problema o della soluzione**, sfruttando l'espressività dei linguaggi visuali.
- **UML come progetto:** Diagrammi di progetto abbastanza dettagliati che vengono utilizzati per:
  1. **Il reverse engineering**, ovvero per visualizzare e comprendere meglio **del codice già esistente** mediante dei diagrammi UML. In questo caso, uno strumento UML legge il codice sorgente o binario per **generare** (di solito) **dei diagrammi UML dei package, delle classi e di sequenza**. Questi "progetti" possono aiutare il lettore a capire i principali elementi, le strutture e le collaborazioni

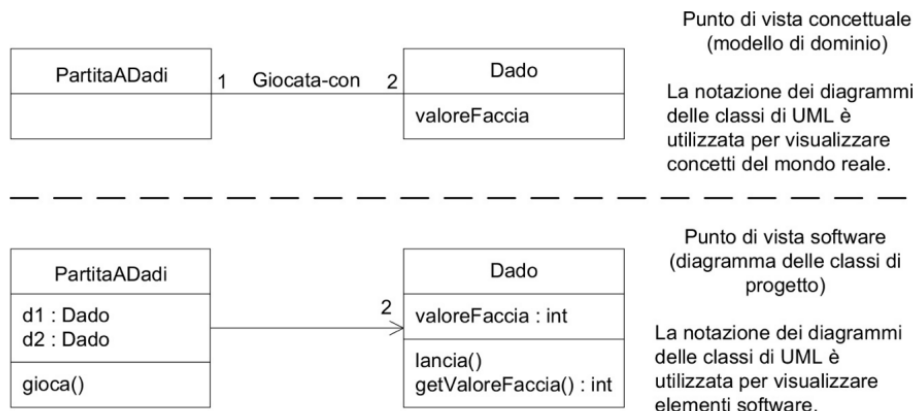
2. **Il forward engineering**, ovvero per la **generazione di codice**. In questo caso, alcuni diagrammi dettagliati possono fornire una **guida alla generazione di codice** da fare manualmente o automaticamente con uno strumento. Solitamente, i diagrammi sono utilizzati per **specificare una parte di codice**, mentre il resto del codice viene scritto da uno sviluppatore durante la codifica, magari applicando UML come abbozzo.
- **UML come linguaggio di programmazione**: La specifica **completamente eseguibile** di un sistema software con UML. Il codice viene generato **automaticamente** e non viene normalmente né visto né modificato dagli sviluppatori; quindi UML viene usato come vero e proprio **linguaggio di programmazione**. Questo utilizzo di UML richiede un modo **pratico** per rappresentare sotto forma di diagrammi **tutto il comportamento o la logica** (probabilmente tramite diagrammi di interazione e di stato). Si tratta di un approccio **ancora in corso di sviluppo** sia in termini di teoria sia in termini di usabilità e robustezza degli strumenti.

La **modellazione agile** enfatizza l'uso di UML come **abbozzo**; si tratta di un metodo comune per applicare UML, spesso con un elevato ritorno in termini di **investimento di tempo** (che è normalmente breve).

### 1.5.3 Due punti di vista per applicare UML

UML descrive dei tipi **grezzi** di diagrammi, come i diagrammi delle classi e i diagrammi di sequenza; tuttavia UML **non impone un particolare punto di vista di modellazione per l'uso di questi diagrammi**; quindi la stessa notazione può essere usata secondo **due punti di vista** (o prospettive) e **tipi di modelli**:

- **Punto di vista concettuale**: I diagrammi sono scritti e interpretati **come descrizioni di oggetti del mondo reale** o nel dominio di interesse
- **Punto di vista software**: I diagrammi, che utilizzano **la stessa notazione del punto di vista concettuale**, descrivono astrazioni o componenti software. In particolare, i diagrammi possono descrivere:
  1. **Implementazioni software** con riferimento a una particolare tecnologia
  2. **Specifiche e interfacce** di componenti software, ma **indipendentemente** da ogni possibile implementazione



Quindi, in pratica, UML viene usato:

1. **Nell'analisi**, principalmente secondo il **punto di vista concettuale**
2. **Nella progettazione**, principalmente secondo il **punto di vista software**

#### 1.5.4 Significato di classe

Nell'UML grezzo, abbiamo chiamato "classi" un insieme di oggetti; ma questo termine racchiude una **varietà di casi**: oggetti fisici, concetti astratti, elementi software, eventi e così via. In particolare, una classe UML è un caso particolare di un modello UML generale chiamato **classificatore**, che è qualcosa che ha delle caratteristiche strutturali e/o comportamentali e comprende **classi, attori, interfacce e casi d'uso**. Un metodo **impone una terminologia alternativa sovrapposta all'UML grezzo**; in particolare, ci adegueremo a quella di **UP** (Unified Process), che chiama:

- **Classe concettuale**: Oggetto o concetto **del mondo reale** da un punto di vista **concettuale**. Il modello di dominio di UP contiene **classi concettuali**
- **Classi software**: Una classe che rappresenta un **componente software**, da un punto di vista **software**, indipendentemente dal processo, metodo o linguaggio di programmazione. Il modello di progetto di UP contiene **classi software**.

#### 1.5.5 Vantaggi della modellazione visuale

Disegnare e leggere UML implica che si sta lavorando in **modo visuale**. La modellazione visuale ci permette di sfruttare le capacità del nostro cervello di **comprendere rapidamente simboli, unità e relazioni nelle notazioni** (prevalentemente bidimensionali) a "rettangoli e linee". I diagrammi ci aiutano a vedere o esaminare meglio il **quadro generale** e le relazione tra elementi dell'analisi del software e allo stesso tempo ci permettono di **ignorare o nascondere i dettagli poco interessanti**.

## 2 Processi per lo sviluppo del software

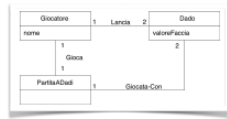
Un **processo per lo sviluppo del software** (o **processo software**) definisce un approccio disciplinato per la **costruzione**, il **rilascio** e la **manutenzione del software**. Definisce quindi **chi fa che cosa, quando e come** per raggiungere un certo obiettivo. In particolare:

- **Cosa** sono le **attività**
- **Chi** sono i **ruoli**
- **Come** sono le **metodologie**
- **Quando** riguarda l'**organizzazione temporale** delle attività

Le attività fondamentali di un processo di sviluppo sono:



Requisiti



Analisi



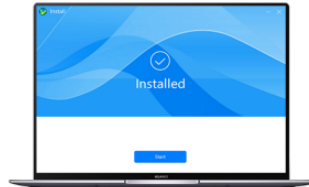
Progettazione



Implementazione



Validazione



Rilascio e installazione



Manutenzione  
Ed Evoluzione



Gestione  
del Progetto