



## BugBusters

Email: [bugbusters.unipd@gmail.com](mailto:bugbusters.unipd@gmail.com)

Gruppo: 4

Università degli Studi di Padova

Laurea in Informatica

Corso: Ingegneria del Software

Anno Accademico: 2025/2026

# Glossario

Versione 0.0.3

<b>Stato</b>	In redazione
<b>Redattori</b>	Alberto Autiero
<b>Verificatori</b>	[Nome Cognome]
<b>Uso</b>	Interno ed esterno
<b>Destinatari</b>	Prof. Tullio Vardanega, Prof. Riccardo Cardin, BugBusters, Eggon

## Indice

Introduzione	8
A	9
AI (Artificial Intelligence) . . . . .	9
AI Co-Pilot . . . . .	9
AI generativa . . . . .	9
Accoppiamento . . . . .	9
Accoppiamento (Coupling) . . . . .	9
Affidabilità (Reliability) . . . . .	9
Agile . . . . .	9
Aggregazione . . . . .	9
Amministratore . . . . .	10
Analisi dei Requisiti . . . . .	10
Analista . . . . .	10
API (Application Programming Interface) . . . . .	10
Approvazione . . . . .	10
Architettura a livelli . . . . .	10
Architettura dei microservizi . . . . .	10
Architettura multilivello . . . . .	10
Architettura Pipe-and-Filter . . . . .	10
Architettura Three-Tier . . . . .	11
Associazione . . . . .	11
Attore . . . . .	11
Attore principale . . . . .	11
Attore secondario . . . . .	11
Availability (Disponibilità) . . . . .	11
AWS (Amazon Web Services) . . . . .	11
B	12
Back-end . . . . .	12
Baseline . . . . .	12
Best Practice . . . . .	12
Builder . . . . .	12
C	13
Capitolato . . . . .	13
Casi d'uso . . . . .	13
Cedolini . . . . .	13
Cedolini Massivi . . . . .	13
Cerimonia . . . . .	13
Committente . . . . .	13

Composizione . . . . .	13
Comprensibilità . . . . .	13
Constructor Injection . . . . .	13
Controllo di Qualità . . . . .	14
Cruscotto/Dashboard . . . . .	14
Cruscotto di Controllo . . . . .	14
<b>D</b>	15
Decomposizione funzionale . . . . .	15
Decisione esterna . . . . .	15
Decisione interna . . . . .	15
Dependency Injection . . . . .	15
Design Pattern . . . . .	15
Design pattern architetturali . . . . .	15
Diagramma dei componenti . . . . .	15
Diagramma dei casi d'uso . . . . .	15
Diagramma delle classi . . . . .	15
Diagramma degli stati . . . . .	16
Diagramma di attività . . . . .	16
Diagramma di deployment . . . . .	16
Diagramma di Gantt . . . . .	16
Diagramma di PERT . . . . .	16
Diagramma di sequenza . . . . .	16
Dipendenza . . . . .	16
Discord . . . . .	16
Dispatch . . . . .	16
Dominio d'uso . . . . .	17
<b>E</b>	18
Economicità . . . . .	18
Efficacia . . . . .	18
Efficienza . . . . .	18
Entity Resolution . . . . .	18
Error Handling . . . . .	18
Ereditarietà . . . . .	18
Ereditarietà di classe . . . . .	18
Ereditarietà dell'interfaccia . . . . .	18
<b>F</b>	19
Flessibilità . . . . .	19
Front-end . . . . .	19
Funzionalità . . . . .	19

G	20
GDPR (General Data Protection Regulation) . . . . .	20
Gestione dei rischi . . . . .	20
GitHub . . . . .	20
Google Meet . . . . .	20
Glossario . . . . .	20
H	21
Human in the loop . . . . .	21
I	22
Implementazione . . . . .	22
Incapsulamento . . . . .	22
Incapsulamento (Encapsulation) . . . . .	22
Information Hiding . . . . .	22
Interfaccia . . . . .	22
Issue . . . . .	22
K	23
KPI (Key Performance Indicator) . . . . .	23
L	24
Latex . . . . .	24
Livelli aperti . . . . .	24
Livelli chiusi . . . . .	24
LLM (Large Language Model) . . . . .	24
Load balancer . . . . .	24
M	25
Microservizio . . . . .	25
Microsoft Teams . . . . .	25
Model . . . . .	25
Model-View-Controller (MVC) . . . . .	25
Model-View-Presenter (MVP) . . . . .	25
Model-View-ViewModel (MVVM) . . . . .	25
Modularità . . . . .	25
O	26
OCR (Optical Character Recognition) . . . . .	26
P	27
Pattern Creazionali . . . . .	27
Piano di Qualifica . . . . .	27
Polimorfismo . . . . .	27
Post-condizione . . . . .	27
Pre-condizione . . . . .	27

Preventivo . . . . .	27
Procedimento agile . . . . .	27
Procedimento bottom-up . . . . .	27
Procedimento top-down . . . . .	28
Prodotto (Product) . . . . .	28
Progettazione di dettaglio . . . . .	28
Progettazione logica . . . . .	28
Progettista . . . . .	28
Programmatore . . . . .	28
Progetto . . . . .	28
Prompt . . . . .	28
Proponente . . . . .	28
Processi organizzativi . . . . .	29
Processi primari . . . . .	29
Processi di supporto . . . . .	29
Pull Model . . . . .	29
Push Model . . . . .	29
PWA (Progressive Web App) . . . . .	29
<b>Q</b>	<b>30</b>
Qualità . . . . .	30
Qualità architetturale . . . . .	30
<b>R</b>	<b>31</b>
Rating System . . . . .	31
RBAC (Role-Based Access Control) . . . . .	31
Redattore . . . . .	31
Repository . . . . .	31
Requisito . . . . .	31
Requisiti desiderabili . . . . .	31
Requisiti funzionali . . . . .	31
Requisiti non funzionali . . . . .	31
Requisiti obbligatori . . . . .	31
Requisiti opzionali . . . . .	32
Requirements Baseline . . . . .	32
Requisito software . . . . .	32
Requisito utente . . . . .	32
Responsabile . . . . .	32
Riusabilità . . . . .	32
Robustezza . . . . .	32
<b>S</b>	<b>33</b>

Sandbox di Sviluppo . . . . .	33
Scalabilità . . . . .	33
Scenario . . . . .	33
Scenario alternativo . . . . .	33
Scenario principale . . . . .	33
Scope . . . . .	33
SCRUM . . . . .	33
SEMAT . . . . .	33
Sottotipizzazione . . . . .	33
Specification Tecnica . . . . .	34
Sprint . . . . .	34
Stakeholder . . . . .	34
Stima dei costi . . . . .	34
<b>T</b>	<b>35</b>
Technology Baseline . . . . .	35
Template . . . . .	35
Test . . . . .	35
Transazioni distribuite . . . . .	35
<b>U</b>	<b>36</b>
Unità architetturali . . . . .	36
<b>V</b>	<b>37</b>
Validatore . . . . .	37
Validazione . . . . .	37
Verifica . . . . .	37
Verifica vs. Validazione . . . . .	37
Verificatore . . . . .	37
Versionamento/versioning . . . . .	37
ViewModel . . . . .	37
<b>W</b>	<b>38</b>
Way of Working . . . . .	38
Workflow . . . . .	38

## Versioni del documento

Versione	Data	Descrizione	Redatto	Verificatori	Approvato
0.0.3	23/11/2025	Sono stati aggiunti nuovi termini	Alberto Autiero	-	-
0.0.2	17/11/2025	Sono stati aggiunti nuovi termini	Alberto Autiero	-	-
0.0.1	06/11/2025	Prima stesura del documento con i termini del capitolo aggiudicato (5) e diapositive dei docenti	Alberto Autiero	-	-

## Introduzione

Questo documento è stato redatto per garantire chiarezza e uniformità nell'interpretazione della terminologia utilizzata nell'intera documentazione del progetto. Al fine di eliminare possibili fraintendimenti, vengono presentate qui le definizioni dei termini tecnici, degli acronimi e delle espressioni che potrebbero risultare ambigue. L'obiettivo del glossario consiste nell'assicurare che tutti i membri del gruppo di lavoro e i destinatari della documentazione condividano un'interpretazione univoca e coerente della terminologia impiegata.

La nomenclatura utilizzata per evidenziare che una parola, con annessa la sua definizione, è presente all'interno del Glossario viene indicata come segue: **parolaG**.

## A

### **AI (Artificial Intelligence)**

Campo dell'informatica che si occupa di sviluppare sistemi in grado di svolgere compiti che normalmente richiedono l'intelligenza umana, come il riconoscimento vocale, la visione artificiale, l'elaborazione del linguaggio naturale e il processo decisionale.

### **AI Co-Pilot**

Modulo di supporto automatizzato per gli studi dei Consulenti del Lavoro (CdL), che assiste nei processi di gestione, riconoscimento e distribuzione documentale attraverso tecniche di intelligenza artificiale.

### **AI generativa**

Tipo di intelligenza artificiale in grado di creare autonomamente nuovi contenuti (testo, immagini, audio, codice) basandosi su pattern appresi dai dati di addestramento.

### **Accoppiamento**

Grado di interdipendenza tra componenti software. Un accoppiamento stretto indica forte dipendenza, mentre un accoppiamento debole minimizza le dipendenze, favorendo manutenibilità e riutilizzo del codice.

### **Accoppiamento (Coupling)**

Grado di interdipendenza tra componenti software. Un accoppiamento stretto indica forte dipendenza, mentre un accoppiamento debole minimizza le dipendenze, favorendo manutenibilità e riutilizzo del codice.

### **Affidabilità (Reliability)**

Capacità di un sistema software di mantenere il proprio livello di prestazioni in condizioni specificate per un determinato periodo di tempo.

### **Agile**

Metodologia di sviluppo del software iterativa e incrementale che si basa su principi come la collaborazione continua con il cliente, la consegna frequente di software funzionante e la capacità di rispondere ai cambiamenti dei requisiti.

### **Aggregazione**

Relazione strutturale in OOP che rappresenta un legame "parte-tutto" dove gli componenti possono esistere indipendentemente dall'oggetto che li contiene. È una forma di associazione con condivisione dei riferimenti.

## **Amministratore**

Figura responsabile della gestione dell'infrastruttura, degli strumenti di sviluppo e dei processi del progetto. Si occupa di configurare e mantenere gli ambienti di lavoro e garantire che il team abbia a disposizione gli strumenti necessari.

## **Analisi dei Requisiti**

Processo sistematico volto a identificare, documentare e validare i bisogni, i vincoli e le esigenze del progetto, trasformandoli in requisiti formali che guideranno lo sviluppo del software.

## **Analista**

Figura professionale specializzata nell'analisi dei requisiti e nella specifica delle funzionalità del sistema. Collabora con gli stakeholder per comprendere le esigenze e tradurle in specifiche tecniche.

## **API (Application Programming Interface)**

Insieme di definizioni, protocolli e strumenti per la costruzione e l'integrazione di software applicativo. Consente a diversi sistemi di comunicare tra loro.

## **Approvazione**

Processo formale attraverso il quale un documento o una funzionalità viene accettata e considerata completa dopo aver superato le verifiche e validazioni necessarie.

## **Architettura a livelli**

Pattern architettonico che organizza il sistema in livelli gerarchici, onde ogni livello fornisce servizi al livello superiore e utilizza servizi del livello inferiore.

## **Architettura dei microservizi**

Approccio architettonico che struttura un'applicazione come una collezione di servizi piccoli, autonomi e indipendenti, ciascuno responsabile di una specifica funzionalità di business.

## **Architettura multilivello**

Pattern architettonico che separa le responsabilità dell'applicazione in livelli logici distinti, tipicamente presentazione, logica di business e persistenza dati.

## **Architettura Pipe-and-Filter**

Pattern architettonico in cui i componenti (filtri) elaborano i dati in sequenza, passandoli attraverso connessioni (pipe).

## **Architettura Three-Tier**

Architettura software che divide l'applicazione in tre livelli: presentazione, logica di business e dati.

## **Associazione**

Relazione strutturale in OOP che rappresenta una connessione duratura tra oggetti di classi diverse, dove gli oggetti interagiscono per un periodo di tempo prolungato condividendo riferimenti.

## **Attore**

Ruolo svolto da un utente o sistema esterno che interagisce con il sistema software per raggiungere obiettivi specifici. Gli attori possono essere primari (beneficiari diretti) o secondari (fornitori di servizi).

### **Attore principale**

Utente o sistema esterno che interagisce direttamente con il sistema software per raggiungere un obiettivo specifico nel caso d'uso. È il protagonista dell'interazione e trae beneficio diretto dall'esecuzione del caso d'uso.

### **Attore secondario**

Utente o sistema esterno che fornisce servizi o supporto all'attore principale durante l'esecuzione di un caso d'uso. Partecipa all'interazione ma non è il beneficiario principale del risultato, spesso fornendo funzionalità accessorie o di supporto.

## **Availability (Disponibilità)**

Misura della percentuale di tempo in cui un sistema è operativo e accessibile agli utenti.

## **AWS (Amazon Web Services)**

Piattaforma di cloud computing che offre servizi di calcolo, storage, database e altre funzionalità per supportare lo sviluppo e il deployment di applicazioni.

## B

### **Back-end**

Parte di un'applicazione software che gestisce la logica di business, l'elaborazione dei dati e la comunicazione con il database. Opera sul server ed è inaccessibile direttamente all'utente finale.

### **Baseline**

Versione approvata e formalmente controllata di un documento o di un componente software che serve come riferimento per sviluppi successivi. Le modifiche successive richiedono procedures formali di controllo.

### **Best Practice**

Insieme di tecniche, metodi e procedures che sono state riconosciute come le più efficaci ed efficienti per raggiungere un obiettivo specifico in un determinato contesto.

### **Builder**

Pattern creazionale che separa la costruzione di un oggetto complesso dalla sua rappresentazione, in modo che lo stesso processo di costruzione possa creare diverse rappresentazioni. Il pattern Builder è composto da Director, Builder e Product.

## C

### **Capitolato**

Documento contrattuale che specifica i requisiti, le caratteristiche tecniche e le condizioni di un progetto software proposto da un'azienda proponente per il corso di Ingegneria del Software.

### **Casi d'uso**

Tecnica di specifica dei requisiti che descrive le interazioni tra gli attori (utenti o sistemi esterni) e il sistema software per raggiungere un obiettivo specifico.

### **Cedolini**

Documenti retributivi che attestano la retribuzione corrisposta al dipendente per un determinato periodo di paga.

### **Cedolini Massivi**

File contenenti più documenti retributivi aggregati, da suddividere e assegnare ai singoli destinatari tramite riconoscimento automatico.

### **Cerimonia**

Evento formale o informale nel framework Scrum che segue un agenda prestabilita e ha uno scopo specifico, come la Pianificazione dello Sprint, il Daily Stand-up, la Revisione dello Sprint o la Retrospettiva.

### **Committente**

Soggetto che commissiona il progetto, definendone gli obiettivi, i vincoli e i requisiti, e che ricepisce il prodotto finale. Nel contesto del corso di Ingegneria del Software, può essere un'azienda proponente o un docente.

### **Composizione**

Relazione strutturale in OOP che rappresenta un legame "parte-tutto" forte dove gli componenti non possono esistere indipendentemente dall'oggetto che li contiene. Implica ownership esclusiva e distruzione concomitante.

### **Comprensibilità**

Capacità di un sistema software di essere facilmente compreso dai suoi utilizzatori, riducendo lo sforzo cognitivo necessario per il suo utilizzo.

### **Constructor Injection**

Tecnica di Dependency Injection in cui le dipendenze vengono fornite a un componente attraverso il suo costruttore.

**Controllo di Qualità**

Insieme di attività e tecniche volte a garantire che i prodotti software soddisfino gli standard di qualità prestabiliti. Il controllo di qualità si concentra sull'identificazione di difetti nel prodotto realizzato.

**Cruscotto/Dashboard**

Interfaccia utente che presenta in forma grafica e sintetica le metriche, gli indicatori di performance e lo stato corrente del progetto o dell'applicazione.

**Cruscotto di Controllo**

Strumento di monitoraggio che fornisce una visualizzazione in tempo reale degli indicatori chiave di prestazione (KPI) e delle metriche di qualità del progetto. Consente di tracciare lo stato delle attività di verifica e validazione.

## D

### **Decomposizione funzionale**

Processo di scomposizione di un sistema complesso in funzioni o componenti più piccoli e gestibili.

### **Decisione esterna**

Scelta presa da entità esterne al team di progetto (come il committente o il proponente) che vincola le attività del progetto e che il team deve rispettare.

### **Decisione interna**

Scelta presa dal team di progetto riguardante aspetti tecnici, organizzativi o metodologici, documentata per garantire tracciabilità e coerenza nelle attività successive.

### **Dependency Injection**

Pattern architettonale in cui le dipendenze di un componente vengono fornite dall'esterno, anziché essere create internamente al componente stesso.

### **Design Pattern**

Soluzione progettuale generale e riutilizzabile a un problema ricorrente all'interno di un dato contesto nella progettazione di software. I design pattern non sono progetti finiti che possono essere trasformati direttamente in codice, ma modelli astratti che devono essere adattati al caso specifico.

### **Design pattern architetturali**

Soluzioni progettuali ricorrenti e collaudate per problemi architetturali comuni nei sistemi software.

### **Diagramma dei componenti**

Diagramma UML che mostra l'organizzazione e le dipendenze tra i componenti software di un sistema.

### **Diagramma dei casi d'uso**

Diagramma UML che descrive le interazioni tra gli attori e il sistema, mostrando i diversi scenari possibili per raggiungere un obiettivo specifico.

### **Diagramma delle classi**

Diagramma UML che mostrano le classi del sistema, i loro attributi, metodi e le relazioni tra di esse (associazioni, ereditarietà, dipendenze, ecc.).

**Diagramma degli stati**

Diagramma UML che descrive il comportamento di un oggetto in risposta a eventi, mostrando i diversi stati in cui può trovarsi e le transizioni tra essi.

**Diagramma di attività**

Diagramma UML che modellano il flusso di controllo o il flusso di dati tra attività, utilizzati per descrivere la logica di procedura, di business o di caso d'uso.

**Diagramma di deployment**

Diagramma UML che mostra la configurazione fisica dei nodi di elaborazione e dei componenti software eseguiti su di essi.

**Diagramma di Gantt**

Strumento di pianificazione progettuale che visualizza le attività su un asse temporale, mostrando durate, sequenzialità, parallelismo e progressi rispetto alle pianificazioni.

**Diagramma di PERT**

(Program Evaluation and Review Technique) Strumento di analisi delle dipendenze temporali entre attività di progetto, utilizzato per identificare cammini critici e margini temporali (slack time).

**Diagramma di sequenza**

Diagramma UML che mostra le interazioni tra oggetti in sequenza temporale, evidenziando l'ordine dei messaggi scambiati.

**Dipendenza**

Relazione tra componenti software per cui un componente (dipendente) requiste un altro componente (dipenduto) per funzionare correttamente. Le modifiche al componente dipenduto possono influenzare il componente dipendente.

**Discord**

Piattaforma di comunicazione tramite chat vocale, testuale e video, utilizzata dal gruppo per la comunicazione interna e la collaborazione quotidiana.

**Dispatch**

Processo di distribuzione automatizzata dei documenti verso i destinatari finali attraverso diversi canali (es. app, portale, email, PEC).

**Dominio d'uso**

Contesto specifico o ambiente operativo in cui il sistema software sarà impiegato, comprendente le caratteristiche degli utenti finali, le condizioni operative, i vincoli tecnologici e le regole di business che definiscono l'ambito di applicazione del prodotto.

## E

### **Economicità**

Principio gestionale che combina efficienza ed efficacia, misurando la capacità di raggiungere obiettivi prefissati (efficacia) impiegando le risorse minime indispensabili (efficienza).

### **Efficacia**

Capacità di raggiungere gli obiettivi prefissati e produrre i risultati attesi, indipendentemente dalle risorse impiegate.

### **Efficienza**

Rapporto entre i risultati ottenuti e le risorse impiegate per conseguirli. Un processo è efficiente quando raggiunge i suoi obiettivi utilizzando il minimo di risorse necessarie.

### **Entity Resolution**

Processo di identificazione e associazione di entità (es. persone o aziende) a partire da dati parziali o duplicati, tramite algoritmi di matching e disambiguazione.

### **Error Handling**

Insieme di tecniche e meccanismi di programmazione per gestire le condizioni di errore che possono verificarsi durante l'esecuzione di un software. L'error handling prevede l'identificazione, la segnalazione e il recupero dagli errori, al fine di mantenere la stabilità e l'affidabilità del sistema.

### **Ereditarietà**

Meccanismo della programmazione orientata agli oggetti che permette a una classe (sottoclasse) di acquisire attributi e metodi di un'altra classe (superclasse), favorendo il riutilizzo del codice e le relazioni di generalizzazione.

### **Ereditarietà di classe**

Meccanismo di ereditarietà in cui una classe eredita da un'altra classe (sia classe concreta che astratta).

### **Ereditarietà dell'interfaccia**

Meccanismo di ereditarietà in cui una interfaccia eredita da un'altra interfaccia, oppure una classe implementa un'interfaccia.

**F****Flessibilità**

Capacità di un sistema software di adattarsi a cambiamenti nei requisiti o nell'ambiente operativo con modifiche minime.

**Front-end**

Parte di un'applicazione software con cui l'utente interagisce direttamente, responsabile della presentazione dei dati e dell'acquisizione dell'input dell'utente.

**Funzionalità**

Caratteristica o capacità specifica che un sistema software deve possedere per soddisfare i bisogni degli utenti e gli obiettivi del progetto.

## G

### **GDPR (General Data Protection Regulation)**

Regolamento generale sulla protezione dei dati dell'Unione Europea che stabilisce norme per la protezione e la libera circolazione dei dati personali.

### **Gestione dei rischi**

Processo sistematico di identificazione, analisi, pianificazione e controllo dei rischi di progetto, volto a minimizzare la probabilità di occorrenza e l'impatto degli eventi negativi.

### **GitHub**

Piattaforma di hosting per repository Git che offre strumenti per il version control, la collaborazione e la gestione del ciclo di vita del software.

### **Google Meet**

Piattaforma di videoconferenza sviluppata da Google che consente meeting virtuali, condivisione dello schermo e chat. Utilizzata dal gruppo per le riunioni di coordinamento e le presentazioni.

### **Glossario**

Documento che raccoglie e definisce i termini specifici, gli acronimi e le parole ambigue utilizzati nella documentazione di progetto, con lo scopo di garantire una comprensione univoca della terminologia da parte di tutti i membri del team e dei destinatari.

## H

### **Human in the loop**

Approccio in cui l'intelligenza artificiale e gli esseri umani collaborano, con l'uomo che supervisiona, corregge o fornisce feedback al sistema AI, particolarmente utile quando la confidenza del sistema è bassa.

# I

## Implementazione

Processo di traduzione di un design in codice eseguibile, realizzando le specifiche funzionali e non funzionali.

## Incapsulamento

Principio della programmazione orientata agli oggetti che consiste nel racchiudere in un'unica entità (classe) dati e metodi che operano su di essi, nascondendo i dettagli implementativi all'esterno.

## Incapsulamento (Encapsulation)

Principio della programmazione orientata agli oggetti che consiste nel racchiudere in un'unica entità (classe) dati e metodi che operano su di essi, nascondendo i dettagli implementativi all'esterno e esponendo solo un'interfaccia pubblica.

## Information Hiding

Principio di progettazione software che consiste nel nascondere i dettagli implementativi di un modulo, esponendo solo le interfacce necessarie, per ridurre l'accoppiamento e aumentare la manutenibilità.

## Interfaccia

In programmazione orientata agli oggetti, un'interfaccia è un tipo astratto che contiene una serie di metodi dichiarati senza implementazione. Le classi che implementano un'interfaccia devono fornire un'implementazione per tutti i suoi metodi. Le interfacce definiscono un contratto che le classi devono seguire.

## Issue

Segnalazione di un problema, un bug o una richiesta di miglioramento nel sistema di tracking del progetto. Ogni issue viene tracciata, assegnata e gestita fino alla risoluzione.

## K

### **KPI (Key Performance Indicator)**

Indicatore chiave di performance che misura l'efficacia di un processo, un'attività o un'organizzazione nel raggiungere i propri obiettivi.

## L

### **Latex**

Sistema di composizione tipografica utilizzato per la produzione di documentazione tecnica e scientifica di alta qualità, particolarmente adatto per documenti complessi con formule matematiche.

### **Livelli aperti**

Architettura a livelli in cui un livello può utilizzare servizi di qualsiasi livello sottostante, non solo di quello immediatamente inferiore.

### **Livelli chiusi**

Architettura a livelli in cui un livello può utilizzare solo i servizi del livello immediatamente inferiore.

### **LLM (Large Language Model)**

Modello di linguaggio di grandi dimensioni addestrato su vastos corpus di testo, in grado di generare, comprendere e elaborare linguaggio naturale in modo sofisticato.

### **Load balancer**

Componente che distribuisce il carico di lavoro tra più server per migliorare le prestazioni e l'affidabilità del sistema.

## M

### **Microservizio**

Approccio architetturale in cui un'applicazione è composta da piccoli servizi indipendenti, ciascuno eseguibile autonomamente e comunicante tramite API.

### **Microsoft Teams**

Piattaforma di collaborazione sviluppata da Microsoft che offre chat, videoconferenze, archiviazione file e integrazione con applicazioni. Utilizzata dal gruppo per le riunioni di coordinamento e la condivisione di documenti.

### **Model**

Componente in un pattern architetturale che rappresenta i dati e la logica di business dell'applicazione, indipendentemente dall'interfaccia utente.

### **Model-View-Controller (MVC)**

Pattern architetturale che separa l'applicazione in tre componenti interconnessi: Model (dati e logica di business), View (interfaccia utente) e Controller (gestisce l'input dell'utente e media tra Model e View).

### **Model-View-Presenter (MVP)**

Pattern architetturale derivato da MVC che separa l'applicazione in Model, View e Presenter, dove il Presenter contiene la logica di presentazione e agisce da intermediario entre View e Model.

### **Model-View-ViewModel (MVVM)**

Pattern architetturale che separa l'applicazione in Model, View e ViewModel, utilizzando il data binding per sincronizzare automaticamente View e ViewModel.

### **Modularità**

Grado in cui un sistema è composto da componenti separati che possono essere sviluppati, testati e mantenuti indipendentemente.

## O

### **OCR (Optical Character Recognition)**

Tecnologia che converte immagini di testo scritto o stampato in testo digitale machine-readable.

## P

### **Pattern Creazionali**

Categoria di design pattern che si occupano dei meccanismi di creazione degli oggetti, cercando di creare oggetti in modo adatto alla situazione. Esempi includono Factory Method, Abstract Factory, Builder, Singleton, etc.

### **Piano di Qualifica**

Documento che definisce le modalità, le risorse e le tempistiche per le attività di verifica e validazione del software. Descrive gli standard di qualità da adottare, le metriche per la misurazione della qualità, i processi di controllo e le responsabilità del team per garantire la qualità del prodotto.

### **Polimorfismo**

Principio della programmazione orientata agli oggetti che permette a oggetti di classi diverse di rispondere allo stesso messaggio (metodo) in modo specifico per la propria classe, favorendo flessibilità ed estensibilità del codice.

### **Post-condizione**

Condizione o stato del sistema che deve essere vero dopo il completamento di un caso d'uso. Definisce il risultato atteso e le garanzie che il sistema fornisce al termine dell'esecuzione del caso d'uso.

### **Pre-condizione**

Condizione o stato del sistema che deve essere vero prima che un caso d'uso possa iniziare. Definisce i prerequisiti necessari per l'esecuzione corretta del caso d'uso.

### **Preventivo**

Documento di pianificazione che stima i costi e le risorse necessarie per lo svolgimento del progetto, basato sulle attività pianificate e sulle risorse disponibili.

### **Procedimento agile**

Approccio iterativo e incrementale allo sviluppo software che enfatizza la flessibilità e la collaborazione con il cliente.

### **Procedimento bottom-up**

Approccio allo sviluppo software che parte dai componenti di basso livello per costruire gradualmente sistemi più complessi.

## **Procedimento top-down**

Approccio allo sviluppo software che parte da una visione d'insieme per scomporre gradualmente il sistema in componenti più piccoli.

## **Prodotto (Product)**

Nel contesto dei design pattern creazionali, il Product è l'oggetto complesso che viene costruito. In particolare, nel pattern Builder, il Product è l'oggetto finale che viene assemblato dal Director utilizzando il Builder.

## **Progettazione di dettaglio**

Fase della progettazione software in cui si specificano nel dettaglio i componenti, le interfacce e le relazioni tra di essi.

## **Progettazione logica**

Fase della progettazione software in cui si definisce l'architettura del sistema senza considerare i dettagli implementativi specifici.

## **Progettista**

Figura responsabile della progettazione dell'architettura software e delle soluzioni tecniche, garantendo che soddisfino i requisiti e siano realizzabili efficientemente.

## **Programmatore**

Figura che implementa il codice sorgente secondo le specifiche tecniche, seguendo le best practice e gli standard di qualità definiti nel progetto.

## **Progetto**

Insieme di attività che devono raggiungere obiettivi specifici a partire da date specifiche, con un inizio e una fine fissate in calendario, disponendo di risorse limitate (persone, tempo, denaro, strumenti) e consumando risorse nel loro svolgersi.

## **Prompt**

Input (testo e/o altri dati) fornito a un modello di intelligenza artificiale per ottenere una risposta o un output specifico.

## **Proponente**

Azienda o organizzazione che propone un capitolato d'appalto per il progetto del corso di Ingegneria del Software, definendone requisiti e obiettivi.

**Processi organizzativi**

Processos trasversali rispetto ai singoli progetti che riguardano la gestione dei processes, delle infrastrutture, del miglioramento continuo e della formazione del personale nell'organizzazione.

**Processi primari**

Processos fondamentali che agiscono direttamente sul ciclo di vita del prodotto software, includendo acquisizione, fornitura, sviluppo, operazione e manutenzione.

**Processi di supporto**

Processos che supportano i processi primari, includendo documentazione, gestione della configurazione, accertamento della qualità, verifica, validazione e risoluzione dei problemi.

**Pull Model**

Pattern architetturale in cui il client richiede attivamente i dati al server quando necessario.

**Push Model**

Pattern architetturale in cui il server invia automaticamente i dati al client senza che quest'ultimo li abbia esplicitamente richiesti.

**PWA (Progressive Web App)**

Applicazione web che utilizza tecnologie web moderne per offrire un'esperienza simile a quella di un'app nativa, funzionando offline e potendo essere installata sul dispositivo.

## **Q**

### **Qualità**

Insieme delle caratteristiche di un prodotto software che gli conferiscono la capacità di soddisfare le esigenze espresse e implicite degli stakeholder. La qualità del software si misura in base a attributi interni (visibili agli sviluppatori) ed esterni (visibili agli utenti).

### **Qualità architetturale**

Insieme delle caratteristiche che definiscono la bontà di un'architettura software in termini di modularità, riusabilità, manutenibilità e altre proprietà desiderabili.

## R

### **Rating System**

Sistema di valutazione che assegna un punteggio o un giudizio a un'entità (es. prodotto, servizio, contenuto) in base a criteri prestabiliti.

### **RBAC (Role-Based Access Control)**

Modello di controllo degli accessi in cui i permessi sono assegnati a ruoli specifici, e gli utenti ottengono i permessi attraverso l'assegnazione a questi ruoli.

### **Redattore**

Membro del team responsabile della stesura e della produzione dei documenti di progetto, garantendo chiarezza, completezza e conformità alle norme stabilite.

### **Repository**

Archivio centrale in cui vengono memorizzati e versionati i file sorgente, la documentazione e le risorse del progetto utilizzando un sistema di controllo versione.

### **Requisito**

Condizione o capacità che deve essere posseduta da un sistema o componente software per soddisfare un contratto, standard, specifica o altro documento formalmente imposto.

### **Requisiti desiderabili**

Requisiti che sono importanti ma non essenziali per il funzionamento base del sistema. La loro implementazione apporta valore aggiunto ma la loro assenza non compromette il progetto.

### **Requisiti funzionali**

Specificano cosa il sistema deve fare, descrivendo le funzionalità, i comportamenti e le interazioni che il software deve supportare.

### **Requisiti non funzionali**

Definiscono come il sistema deve comportarsi in termini di prestazioni, sicurezza, affidabilità, usabilità e altri attributi di qualità, senza riguardo alle funzionalità specifiche.

### **Requisiti obbligatori**

Requisiti che devono essere necessariamente soddisfatti e la cui mancata implementazione comporterebbe il fallimento del progetto. Sono critici per il successo del sistema.

## **Requisiti opzionali**

Requisiti che sono utili ma non necessari, e la cui implementazione dipende dalla disponibilità di risorse e tempo. Possono essere considerati per versioni future del prodotto.

## **Requirements Baseline**

Insieme dei requisiti concordati e formalmente approvati che costituisce il riferimento per lo sviluppo del progetto. Una volta stabilita, qualsiasi modifica alla baseline dei requisiti deve seguire un processo formale di controllo delle modifiche.

## **Requisito software**

Requisito specificato in termini tecnici, destinato agli sviluppatori, che descrive in modo dettagliato e misurabile una funzionalità o un vincolo del sistema software.

## **Requisito utente**

Requisito espresso dal punto di vista dell'utente finale, descritto in linguaggio naturale e senza dettagli tecnici, focalizzato su ciò che l'utente si aspetta che il sistema faccia.

## **Responsabile**

Figura di riferimento del progetto con compiti di coordinamento, pianificazione, gestione delle risorse e comunicazione con docenti e proponenti.

## **Riusabilità**

Capacità di un componente software di essere utilizzato in diversi contesti o applicazioni senza modifiche sostanziali.

## **Robustezza**

Capacità di un sistema software di funzionare correttamente in condizioni anomale o in presenza di input non validi.

## S

### **Sandbox di Sviluppo**

Ambiente isolato per testare e validare funzionalità senza influenzare i sistemi di produzione, utilizzato per sviluppare e verificare nuove feature in sicurezza.

### **Scalabilità**

Capacità di un sistema di gestire un aumento del carico di lavoro aggiungendo risorse, senza modifiche all'architettura.

### **Scenario**

Sequenza di interazioni tra attori e sistema che descrive un percorso specifico attraverso un caso d'uso. Può essere principale (percorso di successo) o alternativo (variazioni ed eccezioni).

### **Scenario alternativo**

Sequenza di interazioni nel caso d'uso che rappresenta un percorso diverso da quello principale, tipicamente gestendo condizioni eccezionali, errori o scelte alternativa dell'utente. Descrive come il sistema reagisce in situazioni non standard.

### **Scenario principale**

Sequenza di interazioni entre l'attore principale e il sistema che descrive il percorso di successo del caso d'uso, dove l'obiettivo viene raggiunto senza intoppi o condizioni eccezionali. Rappresenta il flusso ideale e più frequente di esecuzione.

### **Scope**

Ambito di un progetto, che definisce i confini, i deliverables, gli obiettivi e i compiti che sono inclusi nel progetto.

### **SCRUM**

Framework agile per la gestione dello sviluppo software che enfatizza lo sviluppo iterativo, l'adattamento ai cambiamenti e la consegna incrementale di valore.

### **SEMAT**

(Software Engineering Method and Theory) Iniziativa internazionale per rifondare l'ingegneria del software come disciplina rigorosa, basata su un kernel di elementi essenziali comuni a tutti i metodi di sviluppo software.

### **Sottotipizzazione**

Relazione entre tipi per cui un tipo (sottotipo) può essere utilizzato in ogni contesto in cui è atteso un altro tipo (supertipo), in accordo con il principio di sostituzione di Liskov.

**Specification Tecnica**

Documento che descrive in dettaglio l'architettura, o design e le scelte implementative del sistema software, guidando le attività di sviluppo.

**Sprint**

Periodo di tempo fisso (tipicamente 2-4 settimane) in Scrum durante il quale il team sviluppa e consegna un incremento di prodotto potenzialmente rilasciabile.

**Stakeholder**

Portatore di interesse, ovvero qualsiasi individuo, gruppo o organizzazione che può influenzare o essere influenzato da un progetto, un'azienda o un sistema.

**Stima dei costi**

Processo di previsione dei costi associati alle attività di progetto, considerando risorse umane, strumenti, infrastrutture e altri fattori che influenzano il budget complessivo.

## T

### **Technology Baseline**

Insieme delle tecnologie, framework, librerie e strumenti di sviluppo selezionati e approvati per il progetto. Definisce lo stack tecnologico di riferimento e costituisce la base per le scelte implementative, garantendo coerenza e standardizzazione nell'architettura software.

### **Template**

Struttura o modello predefinito che definisce il layout e l'aspetto di un'interfaccia utente, separando la presentazione dalla logica di business.

### **Test**

Processo sistematico di verifica che il software soddisfi i requisiti specificati e identifichi difetti, attraverso l'esecuzione controllata di casi di test.

### **Transazioni distribuite**

Transazioni che coinvolgono multiple risorse distribuite in diversi nodi di un sistema, richiedendo meccanismi di coordinamento per garantire atomicità e consistenza.

## U

### **Unità architetturali**

Componenti fondamentali che costituiscono l'architettura di un sistema software, come moduli, componenti, connettori e dati.

## V

### **Validatore**

Membro del team responsabile di eseguire attività de validazione, accertando che il prodotto software soddisfi le effettive esigenze del cliente e gli obiettivi de business.

### **Validazione**

Processo che accerta che il prodotto software sviluppato soddisfi le effettive esigenze del cliente e gli obiettivi di business per i quali è stato realizzato. Risponde alla domanda "Stiamo costruendo il prodotto giusto?" e viene tipicamente effettuata attraverso test di accettazione con il cliente.

### **Verifica**

Processo sistematico che determina se i prodotti di lavoro (documenti, codice, componenti) soddisfano i requisiti e le specifiche definite per loro. Risponde alla domanda "Stiamo costruendo il prodotto nel modo giusto?" e include attività come revisioni, ispezioni e test.

### **Verifica vs. Validazione**

La verifica è il processo di valutazione dei prodotti di lavoro di un progetto per accettare che soddisfino i requisiti specificati e le condizioni imposte all'inizio dello sviluppo. Risponde alla domanda "Stiamo costruendo il prodotto nel modo giusto?". La validazione, invece, accerta che il prodotto finale soddisfi le effettive esigenze del cliente e gli obiettivi di business. Risponde alla domanda "Stiamo costruendo il prodotto giusto?". In sintesi, la verifica si concentra sulla correttezza del processo di sviluppo, mentre la validazione sulla correttezza del prodotto finale.

### **Verificatore**

Membro del team responsabile di controllare che documenti, codice e altri prodotti di lavoro rispettano gli standard de qualità definiti, le norme di progetto e siano privi di errori, incoerenze o ambiguità.

### **Versionamento/versioning**

Sistema per gestire le diverse versioni di file, codice sorgente o documenti, permettendo di tracciare le modifiche, collaborare in team e revertire a versioni precedenti.

### **ViewModel**

Componente nel pattern MVVM che espone i dati e i comandi del Model in modo da essere facilmente legati alla View, spesso implementando notifiche di cambiamento per aggiornare automaticamente la View.

## W

### **Way of Working**

Insieme di processi, metodologie, strumenti e pratiche adottati dal team per organizzare e svolgere le attività di progetto in modo coordinato ed efficiente. Definisce come il team collabora, comunica e gestisce il lavoro quotidiano.

### **Workflow**

Sequenza di attività che definiscono un processo di business, dove compiti, informazioni o documenti passano da un partecipante all’altro secondo regole prestabilite.