alittlebyte.swe@gmail.com

Norme di Progetto

Gruppo	aLittleByte (gruppo 19)
Versione	1.0.0
Redattori	Eleonora Bellet, Diego Belluz, Alex Oloni
Verificatori	V.Y. Kaneda Fernandes, Eleonora Bellet, Angela Maule
Destinatari	Prof. Tullio Vardanega, Prof. Riccardo Cardin
Data	15/06/2026

Registro delle Modifiche

Version	Data	Descrizione	Redattore	Verificatore	Approvatore
1.0.0	28/05/2026	Approvazione versione 1.0.0	-	-	Angelica Gastaldello
0.15.0	27/05/2026	Aggiunta sezione processo di miglioramento continuo	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.14.0	20/05/2026	Aggiunta sezione metriche	Diego Belluz	V.Y. Kaneda Fernandes	-
0.13.0	18/05/2026	Aggiunta sezione qualità del software	Diego Belluz	Eleonora Bellet	-
0.12.0	8/05/2026	Struttura dei diari di bordo	Diego Belluz	Eleonora Bellet	-
0.11.2	7/05/2026	Corretti errori di sintassi	Diego Belluz	V.Y. Kaneda Fernandes	-
0.11.1	22/04/2026	Aggiornate e corrette alcune sezioni delle norme	Alex Oloni	Eleonora Bellet	-
0.11.0	19/04/2026	Tracciamento delle azioni, comunicazione, riunioni	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.10.0	18/04/2026	Processi Organizzativi (Definizione e scopo, gestione degli sprint, ruoli)	Alex Oloni	Eleonora Bellet	-
0.9.0	17/04/2026	Aggiunta sezione Validazione e Processo di Validazione	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.8.0	16/04/2026	Controllo della configurazione, Stato della configurazione, Attività di Verifica, Analisi Statica	Alex Oloni	Eleonora Bellet	-
0.7.0	14/04/2026	Denominazione dei documenti, Produzione e Gestione della configurazione	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.6.0	11/04/2026	Processi di supporto (scopo, strumenti utilizzati, documenti prodotti, struttura dei documenti)	Diego Belluz	Angela Maule	-
0.5.0	9/04/2026	Processi di sviluppo	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.4.0	8/04/2026	Processi di fornitura	Diego Belluz	Angela Maule	-
0.3.0	4/04/2026	Architettura dei processi	Eleonora Bellet	V.Y. Kaneda Fernandes	-
0.2.0	31/03/2026	Scopo del documento e scopo del prodotto	Diego Belluz	Angela Maule	-
0.1.0	31/03/2026	Struttura del documento	Eleonora Bellet	V.Y. Kaneda Fernandes	-

Indice

Indice

1 Introduzione

1.1 Scopo del documento

Il presente documento ha l'obiettivo di definire e regolamentare il *Way of Working (WoW)*[G] adottato dal gruppo *aLittleByte* per la realizzazione del progetto didattico. Al suo interno sono specificate le procedure operative, gli strumenti adottati e le regole di collaborazione interne al team, al fine di garantire un approccio strutturato, efficiente e tracciabile per l'intero ciclo di vita del progetto Nexum[G].

La stesura del seguente documento segue un approccio incrementale[G]: le norme qui descritte verranno costantemente aggiornate e affinate nel tempo, supportando il miglioramento continuo dei processi organizzativi, di sviluppo e di supporto del gruppo.

1.2 Scopo del prodotto

Il prodotto da realizzare consiste in una piattaforma software *stand-alone* sviluppata su richiesta dell'azienda proponente[G] *Eggon*, che si affianca a *Nexum*[G] — piattaforma digitale dedicata alla comunicazione interna e alla gestione HR — comunicando con essa tramite API. Il sistema è architetturalmente indipendente da Nexum[G] e comprende due moduli basati sull'Intelligenza Artificiale[G]:

- **AI Assistant Generativo[G]:** Un modulo dedicato agli utenti amministrativi (dashboard[G]) che permette la creazione autonoma di contenuti aziendali (titolo, testo, immagini di copertina) a partire da un prompt[G], garantendo l'adeguamento automatico del tono e dello stile del messaggio.
- **AI Co-Pilot per Consulenti del Lavoro (CdL)[G]:** Un sistema avanzato in grado di riconoscere automaticamente la tipologia di documenti caricati (es. cedolini, comunicazioni), estrarne i destinatari tramite tecniche di OCR[G] ed **Entity Resolution[G]**, e automatizzare lo smistamento e il dispaccio[G] massivo.

L'obiettivo finale è migliorare l'efficienza dei processi HR e semplificare l'interoperabilità tra le aziende e gli studi dei Consulenti del Lavoro. Per fare ciò il gruppo aLittleByte si impegna a completare il progetto entro la data fissata 10 Settembre 2026, con un costo totale di budget di 12.575 €.

Per ulteriori informazioni si rimanda al documento Analisi dei Requisiti[G].

1.3 Glossario

Al fine di evitare ambiguità e garantire una comprensione uniforme della terminologia utilizzata, è stato redatto un documento esterno denominato *Glossario*[G]. I termini tecnici, gli acronimi e le parole con un significato specifico all'interno del progetto sono contrassegnati nel testo da una "G" in apice (es. parola[G]). La loro definizione completa è consultabile nel *Glossario*[G].

1.4 Riferimenti

1.4.1 Normativi

- **Regolamento del Progetto Didattico:** Anno accademico 2025/2026, Corso di Ingegneria del Software, Università di Padova
  - `https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/PD1.pdf`
  - *Data Ultimo accesso: 17/06/2026*
- **Capitolato[G] d'appalto C5:** Nexum[G] - Eggon S.r.l.
  - `https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C5.pdf`
  - *Data Ultimo accesso: 17/06/2026*

1.4.2 Informativi

- **Glossario[G] v2.0.0:** Documento interno del progetto contenente la definizione dei termini tecnici, di dominio e degli acronimi utilizzati nella documentazione.
  - `https://alittlebyte-19.github.io/Documentazione/RTB/Glossario/Glossario.pdf`
  - *Data Ultimo accesso: 17/06/2026*

2 Architettura dei processi

Per la realizzazione di un prodotto di qualità che soddisfi le esigenze del cliente e degli utenti, il gruppo aLittleByte ha deciso di adottare un'architettura specifica e suddividere i vari processi di costruzione del software in quattro macro-aree:

- Processi primari
- Processi di supporto
- Processi organizzativi
- Validazione

2.1 Processi Primari

In questa sezione viene descritto come nasce il software. I processi primari indicano tutte quelle attività che sono direttamente coinvolte nella creazione del prodotto software, è possibile classificare i processi primari in:

- Processi di fornitura
- Processi di sviluppo

2.1.1 Processi di fornitura

I processi di fornitura comprendono tutte le attività preliminari che permettono di avviare lo sviluppo del progetto, in particolar modo la pianificazione[G]: suddivisione del lavoro in fasi e milestone[G] come *RTB*[G] e *PB*[G], l'analisi: come vengono analizzati i capitolati, la stima: come si stimano vari costi e stime.

2.1.1.1 Scopo e obiettivi Il processo di fornitura definisce le procedure e le attività che il gruppo mette in atto per stabilire un accordo operativo con la proponente[G] (*Eggon*). L'obiettivo è trasformare le necessità espresse nel capitolato[G] C5 in un insieme di requisiti tecnici e vincoli contrattuali ben definiti. In questa fase si stabiliscono le basi metodologiche del *"Way of Working*[G]*"* del team, garantendo che ogni attività sia pianificata, tracciabile e verificabile.

2.1.1.2 Attività Operative Il team si impegna nelle seguenti attività chiave:

- **Analisi e Valutazione del Capitolato[G]:** Studio critico della proposta C5 per identificare rischi tecnologici, opportunità di apprendimento e coerenza con le competenze del team.
- **Stima e Pianificazione[G]:** Definizione del budget (12.575€) e delle scadenze temporali (consegna finale entro il 10 settembre 2026), con la suddivisione del lavoro in cicli iterativi (Sprint[G]).
- **Negoziazione e Definizione dell'MVP[G]:** Collaborazione con *Eggon* per identificare il *Minimum Viable Product*[G], garantendo che le funzionalità core (AI[G] Assistant e Co-Pilot) siano prioritizzate correttamente.
- **Accertamento della Qualità:** Definizione degli obiettivi di qualità per i processi e per il prodotto finale.

2.1.1.3 Documentazione Prodotta (Core della Fornitura) La documentazione non è solo un onere burocratico, ma lo strumento principale per garantire la trasparenza verso i committenti e la coerenza interna. Di seguito sono elencati i documenti redatti e mantenuti dal team:

- **Analisi dei Requisiti[G]:** Documento cardine che formalizza le funzionalità del sistema. Include la modellazione degli attori (Redattore[G], Analista dati[G], Consulente del lavoro[G]), la descrizione dei casi d'uso (UC[G]) e la classificazione dei requisiti funzionali[G], qualitativi e di vincolo.
- **Norme di Progetto[G]:** Definiscono il regolamento interno del gruppo. Contengono gli standard di codifica, le procedure di gestione dei ticket su GitHub[G], le convenzioni di naming e le metodologie di lavoro adottate.
- **Piano di Progetto[G]:** Strumento di gestione manageriale che descrive l'allocazione delle risorse, l'analisi dei rischi[G] e il preventivo[G] economico. Viene aggiornato periodicamente con il consuntivo[G] delle ore per monitorare lo stato del budget.

- **Piano di Qualifica[G]:** Specifica la strategia per garantire che il software sia privo di difetti critici. Definisce le metriche (es. *Code Coverage* $\geq 80\%$) e i test (unità, integrazione, sistema) necessari alla validazione.
- **Glossario[G]:** Raccolta di termini tecnici e acronimi per eliminare ambiguità terminologiche tra sviluppatori, committenti e utenti finali.
- **Specifica Tecnica:** Documento di design che descrive l'architettura di sistema (modello C4[G]), i diagrammi delle classi UML e i design pattern scelti per l'implementazione.
- **Manuale Utente:** Guida operativa destinata agli utenti finali per facilitare l'onboarding sulla piattaforma Nexum[G] e spiegare l'utilizzo degli assistenti AI[G].
- **Lettera di Candidatura e Presentazione:** Documenti formali per l'ufficializzazione della partecipazione all'appalto e per la consegna dei vari rilasci (RTB[G] e PB[G]).
- **Verbali (Interni ed Esterni):** Report che tracciano l'ordine del giorno, le discussioni e le decisioni prese durante le riunioni di team o con i referenti di Eggon.

2.1.1.4 Comunicazione e Strumenti di Supporto

Per la gestione dei processi di fornitura, il gruppo si avvale di:

- **GitHub[G]:** Per il versionamento della documentazione tramite *Pull Request*[G] e per l'integrazione con il codice preesistente.
- **Comunicazione Ufficiale:** L'interfaccia con Eggon avviene tramite incontri bisettimanali e via email/Telegram per comunicazioni rapide.
- **Tracciamento:** Ogni decisione rilevante viene convertita in una *issue*[G] su GitHub[G] per garantirne l'assegnazione e il completamento.

2.1.2 Processi di Sviluppo

Questa sezione codifica l'insieme delle pratiche ingegneristiche e operative indispensabili per costruire materialmente la piattaforma software stand-alone richiesta dal capitolato[G]. L'obiettivo è tradurre i requisiti concordati in un applicativo funzionante e architetturalmente indipendente dal sistema originario (*Nexum*[G]), passando attraverso fasi rigorose di design, programmazione e collaudo.

2.1.2.1 Attività Operative e Ciclo di Vita

Il ciclo di sviluppo per l'integrazione dei modelli di Intelligenza Artificiale[G] richiede una sequenza logica di operazioni ingegneristiche specifiche:

- **Analisi e Modellazione del Dominio:** Studio sistemistico per separare nettamente i flussi di lavoro aziendali (creazione contenuti) da quelli consulenziali (gestione documentale massiva).

- **Progettazione Architetturale:** Definizione di un'infrastruttura *stand-alone* che comunichi in modo asincrono con i moduli esterni[G] di Intelligenza Artificiale[G] (AI Post Generator[G] e AI Analyst[G]), per non bloccare l'interfaccia utente durante le operazioni *time-consuming*.
- **Sviluppo Modulare:** L'applicativo viene frammentato in macro-componenti indipendenti (es. modulo di rating[G]/feedback, modulo di split documentale, dashboard[G] analitica).
- **Validazione "Human-in-the-loop[G]":** Implementazione di flussi di sviluppo che prevedano sempre un passaggio di validazione umana per le operazioni AI[G] a bassa confidenza[G] (es. revisione dei destinatari estratti o modifica del testo generato).
- **Collaudo e Rilascio:** Esecuzione di test formali sulle metriche di qualità (es. tempi di risposta, tasso di successo dell'AI[G]) prima dell'approvazione finale.

2.1.2.2 Analisi e Tracciamento dei Requisiti

Fase nevralgica per la definizione del perimetro del prodotto. Per garantire chiarezza e tracciabilità, il team adotta standard di modellazione e nomenclatura precisi per attori e casi d'uso.

- **Modellazione degli Attori:** Il sistema riconosce due categorie di attori per separare le responsabilità di business da quelle computazionali:
  - *Utenti Operativi (Primari):* Utenti umani che innescano le operazioni (Amministratore[G], Redattore[G], Analista dati[G], Consulente del lavoro[G]).
  - *Moduli Esterni[G] (Secondari):* Entità di sistema chiamate in causa per completare l'operazione (AI Post Generator[G], AI Analyst[G]).
- **Nomenclatura dei Casi d'Uso (Use Case):** Il team adotta una convenzione basata sui prefissi per separare logicamente le aree dell'applicativo:
  - UC-[Numero] identifica i casi d'uso.
  - [Caso Padre].[Sottocaso[G]] indica le scomposizioni logiche (es. *UC-1.1* per un sottocaso[G] di *UC-1*).
- **Struttura Descrittiva:** Ogni caso d'uso[G] documentato deve presentare una scheda strutturata contenente: Attori coinvolti, Trigger, Pre e Post-condizioni, Scenario Principale[G] (sequenza ottimale numerata), Scenario Alternativo.

2.1.2.3 Progettazione Architetturale

La progettazione traduce le specifiche in un'infrastruttura flessibile e scalabile.

- **Disaccoppiamento e Asincronia:** Data la natura *data-driven* e i tempi di latenza[G] dell'AI[G], l'architettura deve garantire l'esecuzione asincrona[G] delle operazioni pesanti (es. elaborazione OCR[G] o entity resolution[G]).
- **Modellazione Visiva:** La struttura del codice viene rappresentata tramite diagrammi UML per la definizione dei casi d'uso e la struttura statica. Si adotta inoltre un approccio a componenti per isolare le funzionalità trasversali (raccolta log, gestione ruoli, telemetria).

2.1.2.4 Codifica

Pattern Architetturali

- **Modularità e Riuso:** Suddivisione del codice in moduli con responsabilità singola (es. libreria dei prompt[G] separata dal motore di generazione). Applicazione di pattern architetturali per evitare dipendenze dirette e favorire la manutenibilità.
- **Gestione degli Errori e Resilienza:** Implementazione di meccanismi robusti per la gestione di anomalie, retry di rete e fallback nel caso in cui i servizi AI[G] esterni non siano temporaneamente disponibili (come mappato negli scenari alternativi dei casi d'uso).
- **Tracciabilità e Qualità:** Versionamento continuo del codice, revisione tra pari (*Code Review*[G]) e test automatizzati per validare non solo le logiche applicative, ma anche l'integrazione corretta dei parametri (tono, stile, metadati[G]) passati ai core di intelligenza artificiale[G].

Standard Stilistici Il codice PHP prodotto deve essere conforme allo standard **PSR-12** (*PHP Standards Recommendations 12*), adottato ufficialmente dalla comunità PHP e seguito nativamente da Laravel[G] 12. PSR-12 definisce regole precise su indentazione (4 spazi), apertura delle parentesi, dichiarazione dei tipi e struttura delle classi, garantendo uniformità in tutto il codebase indipendentemente dal membro che scrive il codice.

Convenzioni di Naming nel Codice Le convenzioni seguono gli standard Laravel[G] e PSR-12:

- **Classi:** PascalCase — es. `UserController`, `DocumentService`
- **Metodi e funzioni:** camelCase — es. `getDocumentById()`, `processInvoice()`
- **Variabili:** camelCase — es. `$userId`, `$documentPath`
- **Costanti:** UPPER_SNAKE_CASE — es. `MAX_RETRY_COUNT`
- **Tabelle e colonne DB:** snake_case, tabelle al plurale — es. `processed_documents`, `document_type`
- **File Blade:** kebab-case — es. `invoice-list.blade.php`
- **Variabili d'ambiente:** UPPER_SNAKE_CASE — es. `MINIO_BUCKET`, `DB_CONNECTION`

Strumenti di Linting e Formattazione

- **Laravel[G] Pint:** Tool ufficiale di formattazione per Laravel[G], basato su PHP CS Fixer. Applica automaticamente PSR-12 al codebase. Deve essere eseguito prima di ogni commit; il codice non conforme non può essere integrato nel branch[G] principale.

- **Larastan:** Estensione di PHPStan ottimizzata per Laravel[G]. Esegue analisi statica[G] del codice per rilevare errori di tipo, chiamate a metodi inesistenti e potenziali bug a runtime, senza eseguire il codice. Il livello minimo di analisi adottato è il **livello 5**.

2.2 Processi di supporto

I processi di supporto comprendono tutte le attività, i metodi e gli strumenti che non sono direttamente legati alla scrittura del codice, ma che risultano indispensabili per sostenere i processi primari di sviluppo e fornitura. Per il nostro team, definire questi processi significa stabilire delle regole di base condivise, necessarie per collaborare in modo efficiente ed evitare disorganizzazione durante le varie fasi del progetto didattico. Ciò ci permette, a livello pratico, di garantire la tracciabilità di ogni decisione tecnica e organizzativa presa dal gruppo e di assicurare che tutto il materiale prodotto (documenti, codice, issue[G]) sia costantemente coerente, aggiornato e facilmente accessibile a tutti i membri del team e al committente[G].

2.2.1 Documentazione

2.2.1.1 Scopo Lo scopo del processo di documentazione è stabilire regole e metodologie pratiche per la creazione, la gestione e l'aggiornamento continuo dei documenti di progetto. Per il nostro gruppo, impegnarci a produrre materiale che si mantenga costantemente chiaro, accessibile e aggiornato ci permette di agevolare e rendere trasparente la comunicazione interna tra i membri del gruppo. Inoltre, una repository[G] ben strutturata facilita notevolmente il confronto e l'allineamento con gli stakeholder[G] esterni e l'azienda *Eggon*. Questo approccio rigoroso garantisce un'accurata tracciabilità delle decisioni organizzative e progettuali prese nel corso dello sviluppo, assicurando al contempo che l'intero progetto venga gestito in piena conformità agli standard di qualità che ci siamo prefissati di rispettare.

2.2.1.2 Strumenti Utilizzati Per gestire in modo efficiente la stesura, la revisione, l'automazione e il versionamento della documentazione, il gruppo *aLittleByte* ha scelto di utilizzare i seguenti strumenti:

- **GitHub[G]:** Rappresenta la nostra piattaforma principale per l'archiviazione e il tracciamento storico dei file. Sfruttiamo le funzionalità di branching e pull request[G] per garantire che ogni modifica venga revisionata in modo asincrono.
- **GitHub[G] Projects:** Utilizzato come strumento principale di project management. Il team gestisce il flusso di lavoro tramite una board Kanban dedicata ("aLittleByte - Dashboard[G]") che permette di visualizzare lo stato di tutte le attività.
- **GitHub[G] Actions e GitHub[G] Pages:** Impiegati per l'automazione (Continuous Integration/Continuous Deployment) e per la pubblicazione automatica e trasparente del sito web documentale.
- **LaTeX[G]:** Scelto per la stesura di tutti i documenti tecnici e formali, garantendo un'impaginazione professionale.

- **Python:** Linguaggio di scripting utilizzato per automatizzare task specifici, come la generazione dinamica della struttura del sito web tramite script proprietari.

2.2.1.3 Documenti Prodotti

I documenti ufficiali previsti per il progetto, i quali saranno costantemente aggiornati e mantenuti sotto controllo di versione all'interno del repository[G], sono i seguenti:

- **Analisi dei requisiti[G]:** Il documento in cui vengono estrapolati, classificati e descritti tutti i requisiti del sistema (funzionali e non), i casi d'uso individuati e i rispettivi criteri per l'accettazione finale.
- **Norme di progetto[G]:** Il documento in esame, avente lo scopo di codificare il nostro Way of Working[G], illustrando nel dettaglio le metodologie operative, gli strumenti scelti e le regole di collaborazione interne al team.
- **Piano di progetto[G]:** Il piano strategico che scandisce le tempistiche di sviluppo, regola l'allocazione delle risorse umane ed economiche e pianifica le varie fasi di avanzamento del lavoro.
- **Piano di qualifica[G]:** Il documento volto a fissare le direttive per l'assicurazione della qualità. Al suo interno vengono delineate le metriche di valutazione e le procedure di testing, con l'obiettivo specifico di garantire una copertura del codice non inferiore all'80%.
- **Glossario[G]:** Uno strumento di supporto testuale che definisce in modo inequivocabile la terminologia specifica, tecnica e di dominio, oltre agli acronimi impiegati in tutta la documentazione.
- **Lettera di Presentazione:** La documentazione formale di accompagnamento, debitamente firmata, necessaria per certificare e presentare ufficialmente il RTB[G] al committente[G].
- **Verbali:** I Verbali (interni ed esterni) tengono traccia degli incontri e delle decisioni sia tra i membri del gruppo e che tra il gruppo e la proponente[G].

2.2.1.4 Struttura dei documenti

Tutti i documenti (eccetto i verbali e i diari di bordo) hanno la seguente struttura generale:

- **Copertina:** Riporta il logo, l'indirizzo email del gruppo, il titolo del documento, il nome del gruppo e i metadati[G] essenziali (redattori, verificatori, destinatari e data).
- **Registro delle modifiche:** Tabella posta all'inizio del file per il tracciamento delle versioni. Per ogni aggiornamento indica: numero di versione, data, descrizione della modifica, redattore[G] e approvatore/verificatore[G].
- **Indice:** Elenco gerarchico dei contenuti con rinvii numerici alle pagine, per favorire una navigazione rapida e mirata.
- **Introduzione:** Contiene lo scopo del documento, glossario[G] e riferimenti.
- **Contenuto:** Corpo centrale del testo organizzato in sezioni e sottosezioni numerate (es. 1, 1.1), redatto in stile chiaro e esecutivo.

2.2.1.5 Struttura dei verbali

- **Copertina:** Riporta gli elementi visivi e identificativi del gruppo, il titolo (specificando se verbale interno[G] o esterno e la data), i ruoli assunti per la stesura (redattore[G] e verificatore[G]) e l'elenco dei destinatari.
- **Indice:** Elenco cliccabile delle sezioni per facilitare la navigazione del documento.
- **Informazioni generali:** Tabella riassuntiva contenente i dettagli logistici dell'incontro: luogo o piattaforma utilizzata (Discord[G]), orario di inizio e fine, e l'elenco puntuale dei partecipanti.
- **Ordine del giorno:** Elenco sintetico degli argomenti pianificati e discussi durante la riunione.
- **Attività svolte:** Resoconto testuale, suddiviso in sottoparagrafi, che descrive in modo oggettivo le discussioni affrontate, le problematiche analizzate e le decisioni prese per ciascun punto all'ordine del giorno.
- **To-Do List:** Sezione conclusiva dedicata al tracciamento delle azioni da intraprendere a seguito della riunione. Per ogni task viene indicato il responsabile[G] e viene inserito il collegamento ipertestuale diretto alla rispettiva *Issue*[G] GitHub[G], garantendo tracciabilità completa tra decisione e attività operativa.

2.2.1.6 Struttura dei diari di bordo I Diari di Bordo sono delle diapositive realizzate per gli incontri settimanali con i docenti per vedere lo stato di avanzamento del lavoro. Il loro scopo principale è condividere rapidamente con i docenti e i colleghi i progressi effettuati e gli ostacoli incontrati dal gruppo *aLittleByte*.

Sono documenti brevi che trattano dei seguenti argomenti:

- **Copertina:** Riporta le informazioni identificative di base, ovvero il logo del team, la data della presentazione, il numero del gruppo e i nomi dei componenti.
- **Cosa abbiamo fatto:** Un riepilogo sintetico e puntato delle attività concluse nell'ultimo periodo.
- **Cosa faremo:** La pianificazione[G] operativa per i giorni successivi, indicando i prossimi *task* da completare.
- **Difficoltà e dubbi:** Una sezione in cui vengono elencati i problemi affrontati dal team e i dubbi da sottoporre all'attenzione del corpo docente.

Questa suddivisione in sezioni permette di ottimizzare i tempi della presentazione, garantendo che i messaggi chiave arrivino in modo chiaro e che ci si possa concentrare immediatamente sulla discussione e risoluzione dei problemi.

2.2.1.7 Denominazione dei documenti

Il gruppo *aLittleByte* ha adottato una rigorosa politica di denominazione univoca per tutta la documentazione. Tale scelta metodologica è volta a preservare l'uniformità del repository[G] e a ottimizzare le procedure di ricerca e consultazione dei file.

- **Verbali:** I verbali hanno la seguente denominazione:

  `<Tipo>_<Data>`

  Dove `<Tipo>` rappresenta:

  - **Verbale_Interno**
  - **Verbale_Esterno**

  Mentre `<Data>` è la data della riunione scritta nel formato `AAAA_MM_GG`. Ad esempio, `Verbale_Interno_2026_04_13.pdf` rappresenta il verbale interno[G] del giorno 13/04/2026.

- **Diari di Bordo:** Per i Diari di Bordo viene adottato il seguente standard di denominazione: `Diario_di_Bordo_AAAA_MM_GG.pdf`. Ad esempio, `Diario_di_Bordo_2026_04_13.pdf` rappresenta il diario di bordo[G] presentato il giorno 13/04/2026.

- **Documenti:** I restanti documenti ufficiali seguono una convenzione di denominazione esplicita basata sul loro contenuto, come ad esempio *Norme di Progetto*[G]*.pdf* o *Analisi dei Requisiti*[G]*.pdf*.

2.2.1.8 Produzione

Il ciclo di vita di ogni documento prodotto dal team è regolato da una procedura strutturata, volta a massimizzare la qualità, l'uniformità e la tracciabilità delle informazioni. Il processo si articola nelle seguenti fasi:

- **Pianificazione[G]:** In corrispondenza delle scadenze di progetto, il gruppo stabilisce quali documenti produrre o aggiornare. Per ogni compito vengono designati un redattore[G] e un verificatore[G], scelti nel rispetto dei ruoli previsti dal *Way of Working*[G] e garantendo l'assenza di conflitti d'interesse.

- **Creazione del branch[G]:** Prima di iniziare la stesura, viene aperto un *branch*[G] dedicato nel repository[G] GitHub[G]. Il gruppo *aLittleByte* ha stabilito di creare un ramo separato per ogni singolo file, inclusi i verbali e i diari di bordo, al fine di garantire una gestione granulare delle modifiche e un isolamento totale del lavoro rispetto al ramo principale.

- **Redazione:** Il redattore[G] incaricato sviluppa il contenuto seguendo i template e le direttive stilistiche definite nelle presenti Norme di Progetto[G]. La scrittura avviene tramite gli strumenti appropriati (ad esempio *LaTeX*[G]), assicurando un output professionale e coerente.

  - **Struttura dei periodi:** semplificare i periodi più densi, evitando frasi eccessivamente lunghe o ricche di incisi e separando le informazioni con funzioni diverse.

  - **Terminologia tecnica:** demandare le definizioni e le spiegazioni dei concetti tecnici al *Glossario*^{G}, evitando di ripeterle all'interno del corpo del documento.
  - **Organizzazione logica:** separare fisicamente e visivamente (ad esempio con paragrafi distinti o elenchi) le parti puramente descrittive dalle parti valutative, dalle motivazioni o dalle conseguenze.

- **Revisione:** Terminata la stesura, il documento viene sottoposto a una verifica asincrona. Il verificatore^{G} analizza il file tramite una *pull request*^{G} su GitHub^{G}, segnalando eventuali correzioni necessarie o suggerimenti direttamente nei commenti della piattaforma.

- **Integrazione e Approvazione:** Una volta superata la revisione e ottenuta l'approvazione formale dal Responsabile^{G}, il contenuto viene integrato nel *branch*^{G} *main* del repository^{G}. In questa fase viene consolidata la versione finale nel registro delle modifiche.

2.2.2 Gestione della configurazione

Il processo di Gestione della Configurazione ha l'obiettivo di tracciare, organizzare e controllare in modo sistematico l'evoluzione di tutti gli artefatti di progetto (codice sorgente, documentazione, script). Adottare questo processo permette al gruppo *aLittleByte* di prevenire conflitti, recuperare versioni precedenti in caso di errore e garantire che ogni rilascio sia coerente e verificabile.

2.2.2.1 Strumenti utilizzati Al fine di implementare correttamente tale processo, il gruppo ha selezionato i seguenti strumenti tecnici:

- **Git^{G}:** Software per il controllo di versione distribuito che permette di monitorare ogni variazione apportata ai file sorgente e alla documentazione.

- **GitHub^{G}:** Servizio di hosting basato su cloud utilizzato per la gestione centralizzata del repository^{G}. Viene impiegato non solo per l'archiviazione, ma anche per la pianificazione^{G} operativa tramite le *Issue*^{G} e per la revisione del lavoro attraverso le *Pull Request*^{G}.

2.2.2.2 Controllo della configurazione Il controllo della configurazione è l'attività preposta alla gestione, tracciabilità e autorizzazione di qualsiasi proposta di modifica del progetto. Per tracciare in modo controllato tutte le modifiche, che vengono approvate o meno, il gruppo *aLittleByte* sfrutta le funzionalità native di GitHub^{G}, strutturando il processo attraverso l'uso rigoroso di *Issue*^{G} e *Pull Request*^{G}:

- **Issue^{G}:** Ogni singola attività (che si tratti della stesura di un documento, dello sviluppo di una funzionalità, della correzione di un errore o anche semplicemente contattare l'azienda) deve essere preventivamente tracciata tramite l'apertura di un'*Issue*^{G}. Quest'ultima viene assegnata a uno o più membri incaricati di portarla a termine. Per facilitarne la classificazione, ogni *Issue*^{G} è identificata da una nomenclatura univoca (ad esempio per la stesura dei documenti viene usato il tag *documentazione*).

- **Pull Request[G]:** Rappresenta il meccanismo esclusivo per l'integrazione di qualsiasi modifica all'interno del repository[G]. Al fine di mantenere un elevato standard qualitativo e prevenire errori, il team ha stabilito che nessun componente è autorizzato a effettuare modifiche dirette sul ramo principale (`main`). Perciò ogni integrazione deve necessariamente passare per una *Pull Request*[G], la quale richiede obbligatoriamente un processo di revisione incrociata e può essere inserita nel main solo a seguito di almeno una revisione positiva da parte di altri membri del gruppo.

2.2.2.3 Registrazione stato di configurazione

Ogni documento presenta un *Registro delle modifiche* (eccetto i verbali) per tenere traccia di tutte le modifiche svolte. Il sistema di versionamento è il seguente:

X.Y.Z

Dove:

- **X:** Subisce un incremento solo quando il file viene approvato dal membro del gruppo incaricato del ruolo di approvatore;
- **Y:** Indica un avanzamento significativo, come l'aggiunta di interi capitoli o moduli software che hanno superato positivamente la fase di verifica.
- **Z:** Rappresenta modifiche di dettaglio, correzioni di refusi o aggiornamenti minimi eseguiti durante la stesura o lo sviluppo.

2.2.2.4 Automazione e Rilascio Continuo (CI/CD)

Per assicurare che la documentazione fornita agli stakeholder[G] sia costantemente aggiornata e pubblica, il gruppo ha implementato un processo di *Continuous Deployment* tramite GitHub[G] Actions. Il workflow[G] principale, nominato `Site Deploy`, regola il ciclo di vita della pubblicazione del sito web e si attiva in automatico ad ogni evento di push o pull request[G] verso il branch[G] `main` (ignorando i file non critici come `README.md` e `.gitignore`).

Il processo automatizzato prevede:

1. Il setup dell'ambiente virtuale e del linguaggio Python.
2. L'esecuzione dello script proprietario `.github/site-src/update_index.py`, che genera dinamicamente la struttura statica del sito (HTML, CSS e asset associati).
3. Il caricamento e rilascio automatico (Deploy) del risultato sull'ambiente `github-pages`.

2.2.2.5 Nomenclatura dei Branch e Alberatura

A differenza dell'utilizzo di identificativi numerici, il gruppo *aLittleByte* adotta una convenzione di nomenclatura descrittiva per i branch[G], in modo da rendere immediatamente esplicito l'oggetto del lavoro. La convenzione principale prevede l'uso dello *snake_ case* (tutto minuscolo con trattini bassi) per i documenti generali (es. `piano_di_progetto`, `norme_di_progetto`, `analisi_dei_requisiti`). Vengono tollerate eccezioni descrittive per i verbali o attività specifiche di sviluppo (es. `VerbaleEsterno_YYYY_MM_DD` o `develop-sito`).

L'alberatura della repository[G] documentale è strutturata in modo logico e gerarchico per separare le fasi del progetto didattico. Le directory principali includono:

- `.github`: Contiene i workflow[G] di automazione e gli script Python necessari al rilascio.
- `candidatura`: Contiene i documenti relativi alla fase di candidatura.
- `RTB`[G]: Contiene la documentazione in sviluppo per la *Requirements and Technology Baseline*[G]. Al suo interno è prevista un'ulteriore categorizzazione rigida, come ad esempio la separazione dei verbali in `RTB/Verbali/Verbali Interni` e `RTB/Verbali/Verbali Esterni`.

2.2.3 Qualifica

2.2.3.1 Verifica Le attività di verifica hanno lo scopo di garantire l'assenza di difetti e la totale aderenza di ogni componente prodotta (sia essa documentale o codice sorgente) rispetto alle specifiche e ai vincoli stabiliti. Applicando i principi dell'Ingegneria del Software, questa pratica metodologica mira a soddisfare la seguente domanda: *"Did I build the system right?"*.

L'intera reportistica riguardante i controlli effettuati, i parametri misurati e gli esiti dei collaudi è raccolta e analizzata all'interno del **Piano di Qualifica**[G].

2.2.3.2 Attività di Verifica La verifica interviene nelle singole fasi del nostro progetto, con lo scopo di assicurarci che il lavoro appena svolto non abbia introdotto errori. In termini pratici, questa attività serve a controllare che il risultato ottenuto alla fine di una determinata fase sia sempre coerente con le informazioni e i requisiti che avevamo in partenza.

In questa prima parte del lavoro, il gruppo *aLittleByte* si è dedicato principalmente alla verifica della documentazione. Tramite revisioni incrociate tra i membri, ci siamo assicurati che i contenuti fossero corretti e che i testi rispettassero le regole grammaticali, sintattiche e di formattazione che ci siamo dati nelle presenti Norme di Progetto[G].

Per quanto riguarda invece la verifica del codice sorgente e del prodotto software, ce ne occuperemo nel dettaglio una volta superata la revisione *RTB*[G]. In ogni caso, tutte le decisioni prese, i metodi di controllo utilizzati e i risultati di queste verifiche verranno sempre riportati in modo esaustivo nel documento **Piano di Qualifica**[G].

2.2.3.3 Analisi Statica L'analisi statica[G] è una tecnica di verifica che viene applicata all'oggetto di studio senza richiederne l'esecuzione. Questo approccio ci permette di individuare anomalie e difetti fin dalle primissime fasi del ciclo di vita del software, ben prima che la scrittura del codice sia ultimata.

Questo tipo di analisi può avvenire tramite metodi formali o, più comunemente nel nostro caso, attraverso tecniche di lettura. Il nostro team adotta principalmente due metodologie:

- **Walkthrough**[G]**:** Consiste in una revisione estensiva e senza preconcetti. Il verificatore[G] analizza il prodotto passo dopo passo, simulandone mentalmente la logica e verificando che le regole prestabilite siano rispettate. È un approccio molto approfondito ma dispendioso in termini di tempo, e richiede un intervento interamente manuale.

- **Inspection:** Si tratta di un controllo mirato e guidato dall'esperienza. Il verificatore[G] sa già quali sono le criticità più comuni e utilizza delle liste di controllo (*checklist*) specifiche per scovarle. Pur essendo meno estesa del *walkthrough*[G], l'ispezione è più rapida e permette di standardizzare la ricerca dei difetti.

Al termine di ogni controllo, le attività svolte e i problemi riscontrati vengono opportunamente documentati. Gli errori vengono discussi in modo costruttivo tra il verificatore[G] e l'autore del documento (o del codice); tuttavia, l'applicazione materiale delle correzioni rimane sempre di competenza dell'autore originale, al fine di garantire coerenza e responsabilità. Tutto ciò viene documentato all'interno della *Pull Request*[G], che in caso di errori non viene approvata, e nel relativo documento.

Analisi Statica Automatica In affiancamento alle tecniche manuali, il team adotta **Larastan** (PHPStan per Laravel[G]) come strumento di analisi statica[G] automatica del codice. Larastan viene eseguito al livello di analisi **5** e si occupa di rilevare automaticamente i *code smell* tracciati dalla metrica **MPD12** (es. metodi troppo complessi, variabili non tipizzate, chiamate a metodi inesistenti). L'esecuzione avviene in locale prima di ogni *commit* e nella pipeline di *Continuous Integration*: il codice che non supera l'analisi non può essere integrato nel branch[G] principale.

2.2.3.4 Analisi Dinamica

A differenza della tecnica statica, l'analisi dinamica[G] presuppone l'esecuzione dell'oggetto sottoposto a verifica. Tale approccio esamina le reazioni di specifiche porzioni del software applicandovi un numero definito di "casi prova". Questi test devono possedere i requisiti di ripetibilità e automazione.

Ogni sessione di test è definita dai seguenti stati:

- **NI**: non implementato
- **I**: implementato
- **S**: superato
- **NS**: non superato

2.2.3.5 Test Di Unità

I test d'unità si focalizzano sulla verifica isolata di singole componenti software, come funzioni, classi o metodi, separandole dal resto dell'architettura di sistema.

Il fine di questa attività risiede nel confermare che ogni unità generi i risultati previsti partendo da input determinati. Questo processo consente di intercettare precocemente difetti di implementazione o errori nella logica funzionale durante le fasi iniziali della progettazione.

Ambiente e Procedura Operativa Il framework adottato è **Pest[G] PHP**, incluso nativamente in Laravel[G] 12. I test risiedono nella directory `tests/Unit/` del progetto.

Il programmatore[G] è tenuto a seguire questa procedura prima di aprire una *Pull Request*[G]:

1. Scrivere i test di unità[G] che coprono la logica del codice prodotto.

2. Verificare che tutti i test passino in locale con il comando `./vendor/bin/pest`.

3. Calcolare la *code coverage* con `./vendor/bin/pest --coverage` (richiede il driver **Xdebug** attivo nel container PHP-FPM). La copertura minima richiesta per ogni unità è dell'**80%**, in conformità con la metrica **MPC-CC**.

4. Allegare l'esito del report nella descrizione della *Pull Request*[G].

La pipeline di CI verifica automaticamente questi requisiti a ogni push; una PR con copertura inferiore alla soglia o con test falliti non può essere approvata.

2.2.3.6 Test di Regressione

L'esecuzione dei test di regressione[G] è finalizzata ad accertare che interventi correttivi o evolutivi sul codice non abbiano introdotto nuovi difetti o problemi nelle funzionalità già esistenti.

Non richiedono la scrittura di test ad hoc: sono garantiti dall'esecuzione automatica dell'intera suite Pest[G] PHP (unità + integrazione) nella pipeline di CI ad ogni *push* o apertura di *Pull Request*[G]. Un test precedentemente verde che diventa rosso segnala una regressione e blocca il merge fino alla risoluzione.

2.2.3.7 Test di Integrazione

L'esecuzione dei test d'integrazione è mirata a confermare che le varie componenti del software operino tra loro in modo sinergico e privo di errori.

I test di integrazione[G] sono scritti con **Pest[G] PHP** e risiedono in `tests/Feature/`. A differenza dei test di unità[G], questi test eseguono richieste HTTP reali contro l'applicazione Laravel[G] utilizzando il database PostgreSQL[G] di test (definito in `.env.testing`), verificando l'interazione tra controller, service layer e database. Anche per questi test vale il requisito di copertura minima dell'**80%** e l'obbligo di esecuzione prima di aprire una *Pull Request*[G].

2.2.3.8 Test di Sistema

I test di sistema[G] hanno l'obiettivo di accertare che il comportamento globale del prodotto rispetti integralmente i requisiti software stabiliti durante l'analisi dei requisiti[G].

Vengono eseguiti sull'applicazione deployata nell'ambiente Docker Compose completo (Nginx, PHP-FPM, PostgreSQL[G], Redis[G], MinIO[G]), simulando scenari end-to-end che attraversano tutti i livelli dello stack. Ogni test di sistema[G] è tracciato a un requisito funzionale[G] dell'Analisi dei Requisiti[G] tramite il codice UC[G] corrispondente. I risultati e lo stato di copertura dei requisiti sono riportati nel *Piano di Qualifica*[G].

2.3 Validazione

La validazione rappresenta il processo finale che conferma in modo definitivo che le caratteristiche del prodotto software siano pienamente conformi ai bisogni dell'utilizzo finale e agli obiettivi d'uso previsti. L'obiettivo principale è rispondere al quesito: *"Did I build the right system?"*

Mentre la verifica analizza la correttezza tecnica dell'esecuzione, la validazione si sposta sul piano dell'efficacia, garantendo che le funzionalità sviluppate siano effettivamente idonee a risolvere le problematiche operative individuate e siano allineate con le aspettative del committente[G].

2.3.0.1 Processo di Validazione Il gruppo *aLittleByte* ha provveduto a raccogliere, analizzare e formalizzare tutte le necessità espresse dall'azienda proponente[G] *Eggon* all'interno del documento di *Analisi dei Requisiti*[G]. Questa operazione di classificazione è fondamentale per garantire un tracciamento rigoroso delle specifiche e per predisporre i futuri test di accettazione[G], i quali avranno il compito di accertare che il prodotto software sviluppato rispecchi le richieste del committente[G].

Gli esiti derivanti da queste procedure di validazione saranno riportati e discussi in modo esaustivo nel *Piano di Qualifica*[G], all'interno della specifica sezione dedicata ai Test di Accettazione[G].

Si precisa che, in concomitanza con la pubblicazione dei documenti per l'*RTB*[G], il progetto si trova in una fase antecedente alla codifica e al collaudo del software. Per tale ragione, lo stato di avanzamento dei test riporterà la dicitura temporanea "Non Implementato". I dati e i risultati effettivi dei test verranno integrati in questa sezione solamente in vista della consegna della *PB*[G].

2.4 Processi Organizzativi

2.4.1 Descrizione e scopo

In conformità con le direttive tracciate dallo standard ISO/IEC 12207:1997, il processo di gestione comprende l'insieme delle pratiche e delle metodologie necessarie a governare l'intero ciclo di vita del progetto. Lo scopo primario è assicurare il successo dello sviluppo del prodotto software, garantendo che gli obiettivi prefissati vengano raggiunti nel rigoroso rispetto dei vincoli di costo, delle tempistiche stimate e dei requisiti concordati con il proponente[G].

Per concretizzare tali obiettivi, l'organizzazione del gruppo *aLittleByte* si basa su:

- **Pianificazione[G]:** delle attività da svolgere, della definizione delle scadenze e della suddivisione dettagliata delle mansioni.
- **Monitoraggio e controllo:** Supervisione costante dello stato di avanzamento dei lavori
- **Prevenzione e gestione dei rischi:** Identificazione delle potenziali criticità tecnologiche o organizzative, unita all'adozione di protocolli strutturati per governare le modifiche e per prevenire i possibili imprevisti.
- **Interventi correttivi:** Prontezza e reattività nell'intervenire qualora le metriche di controllo evidenzino deviazioni significative rispetto al Piano di Progetto[G].
- **Organizzazione del team:** Assegnazione trasparente di ruoli e responsabilità ai membri del gruppo, garantendo al contempo il mantenimento di canali di comunicazione efficaci.

In definitiva, l'applicazione metodica di questi processi garantisce al team di mantenere un progetto solido e tracciabile, costituendo il fondamento necessario per assumere decisioni strategiche in qualsiasi momento dello sviluppo.

2.4.2 Gestione degli sprint

Come già citato in precedenza, il gruppo ha deciso di adottare un approccio iterativo, suddividendo il ciclo di vita del progetto in sprint[G] della durata fissa di due settimane. In concomitanza con l'avvio di ogni nuovo sprint[G], il team provvede alla rotazione e all'assegnazione dei ruoli che ciascun membro dovrà ricoprire per l'intera durata dello sprint[G].

L'organizzazione di ogni singola iterazione è scandita da una riunione, organizzata nel seguente modo:

- **Pianificazione[G]:** Il team seleziona gli obiettivi operativi da raggiungere, scompone il lavoro in *task* gestibili e li assegna ai vari componenti, cercando di bilanciare equamente i carichi di lavoro.
- **Sprint[G] Review:** Si analizza lo sprint[G] appena concluso per la verifica concreta dei risultati prodotti. Costituisce l'occasione per valutare il lavoro completato, discutere le criticità tecniche emerse e misurare l'avanzamento effettivo rispetto al piano generale. In questa sede, il team deve obbligatoriamente confrontare le ore preventivate con le ore effettivamente consuntivate: gli scostamenti principali (sottostime, attività impreviste, dipendenze esterne) devono essere motivati e registrati, al fine di aggiornare sistematicamente le previsioni di costo a finire (EAC).
- **Sprint[G] Retrospective:** Un momento di auto-valutazione finale, essenziale per il miglioramento continuo del gruppo. Il team si confronta in modo trasparente sulle dinamiche relazionali e metodologiche, individuando i punti di forza da mantenere e le inefficienze organizzative da correggere in vista del ciclo successivo. In questa fase è obbligatorio revisionare e aggiornare il Registro dei Rischi: per ogni criticità emersa o rischio[G] inatteso manifestatosi durante lo sprint[G], il gruppo deve documentare formalmente nel registro la causa scatenante, l'impatto effettivo sul progetto e l'azione di mitigazione[G] prevista e assegnare.

2.4.2.1 Gestione del Flusso di Lavoro tramite Kanban

Per garantire un controllo visivo e immediato sullo stato di avanzamento delle singole Issue[G] associate al progetto, il gruppo gestisce i task tramite la board GitHub[G] Projects "aLittleByte - Dashboard[G]". La bacheca è modellata secondo i principi Kanban e si articola nelle seguenti colonne operative:

- **Todo:** Task selezionati e pronti per essere affrontati nel ciclo di lavoro corrente.
- **In Progress:** Attività su cui il team sta lavorando attivamente (le issue[G] sono formalmente assegnate ai membri tramite il campo *Assignee*).
- **In Review:** Task completati nello sviluppo o nella stesura e attualmente in attesa di *Peer Review* tramite l'approvazione della relativa *Pull Request*[G].
- **Done:** Lavoro verificato, approvato e unito definitivamente al branch[G] principale.

2.4.3 Ruoli

2.4.3.1 Responsabile Il membro che assume il ruolo di Responsabile[G] agisce come vero e proprio *Project Manager* del gruppo. Ha la responsabilità di coordinare il lavoro del team, pianificare le iterazioni di sviluppo, ottimizzare l'uso delle risorse e fungere da referente principale per le comunicazioni ufficiali con l'azienda proponente[G] e il corpo docente.

Dal punto di vista strettamente operativo, il Responsabile[G] si fa carico di distribuire e assegnare i compiti (tramite le *Issue*[G]) ai vari componenti del gruppo. Inoltre, è sua competenza diretta la stesura e il costante aggiornamento dei seguenti documenti:

- **Verbali (Interni ed Esterni)**
- **Diario di Bordo[G]**
- **Piano di Progetto[G]**
- **Piano di Qualifica[G]**

2.4.3.2 Amministratore L'Amministratore[G] rappresenta la figura di supporto metodologico e tecnologico del team. Il suo scopo principale è definire, ottimizzare e mantenere sempre operativi i processi e gli strumenti di lavoro, assicurando che l'intero gruppo possa procedere con le attività in modo fluido e senza impedimenti di natura tecnica o organizzativa.

Oltre a monitorare le scadenze amministrative e a fornire supporto costante ai colleghi, l'Amministratore[G] si fa carico delle seguenti responsabilità operative:

- **Gestione degli ambienti di lavoro**
- **Manutenzione degli strumenti:** Supervisionare e aggiornare periodicamente i software e i canali adottati dal gruppo per la comunicazione e la collaborazione asincrona.
- **Controllo della configurazione:** Garantire la corretta e rigorosa applicazione delle procedure di versionamento per tutti i file e i documenti prodotti dal team.
- **Norme di Progetto[G]:** Curare la stesura, la revisione e il continuo aggiornamento del presente documento (*Norme di Progetto*[G]), assicurandosi che le regole interne e il *Way of Working*[G] siano sempre allineati alle reali necessità del gruppo.

2.4.3.3 Progettista Il Progettista[G] è la mente tecnica dietro alla struttura del prodotto software. Il suo compito principale è ideare l'architettura del sistema, assicurandosi che sia in grado di soddisfare tutti i requisiti (sia funzionali che di qualità) concordati con il proponente[G]. Lavora a stretto contatto con il resto del gruppo per trasformare le richieste astratte in soluzioni pratiche, efficienti e facili da mantenere nel tempo.

Nello specifico, chi ricopre questo ruolo ha la responsabilità di:

- Valutare e decidere quali linguaggi, *framework* e strumenti tecnici sono più idonei per la realizzazione del progetto.

- **Disegnare l'architettura:** Definire come sarà strutturato il sistema a livello generale, individuando i macro-componenti e il modo in cui dovranno comunicare tra loro.

- Supervisionare il lavoro di stesura del codice, assicurandosi che l'implementazione pratica rispetti le specifiche tecniche e non si allontani dall'idea progettuale di partenza.

2.4.3.4 Analista L'Analista[G] rappresenta il punto di incontro tra le esigenze dell'azienda proponente[G] e il lavoro tecnico del team di sviluppo. Il suo compito principale è comprendere a fondo il problema, raccogliere le richieste e trasformarle in regole chiare, complete e ben definite, evitando qualsiasi ambiguità. In pratica, lavora per assicurarsi che tutti sappiano esattamente *cosa* deve essere costruito, prima ancora di decidere *come* farlo.

Nello specifico, chi ricopre questo ruolo si fa carico di:

- **Stesura dell'Analisi dei Requisiti[G]:** Redigere e mantenere costantemente aggiornato il documento che cataloga tutti i casi d'uso e le specifiche del progetto.

- **Gestione dei dubbi e delle modifiche:** Interfacciarsi direttamente con il proponente[G] per risolvere eventuali incertezze, chiedere chiarimenti o concordare variazioni ai requisiti iniziali.

- **Supporto al team:** Collaborare con i progettisti e i programmatori per garantire che le specifiche scritte vengano interpretate nel modo corretto e implementate senza deviare dagli obiettivi concordati.

2.4.3.5 Programmatore Il Programmatore[G] è la figura operativa che materialmente "dà vita" al prodotto software. Il suo compito principale è tradurre l'architettura e le specifiche pensate dal Progettista[G] in codice reale e funzionante. Lavora con l'obiettivo di produrre un software non solo corretto, ma anche pulito, efficiente e facile da mantenere o aggiornare in futuro.

Nello specifico, chi ricopre questo ruolo si fa carico di:

- **Stesura del codice:** Scrivere il codice sorgente rispettando rigorosamente le norme stilistiche e gli standard di qualità definiti dal gruppo.

- **Testing e collaudo preliminare:** Sviluppare ed eseguire i test per verificare che le singole porzioni di codice si comportino come previsto e garantiscano un'elevata affidabilità.

- **Risoluzione dei problemi:** Lavorare a stretto contatto con il resto del team per individuare i difetti, correggere i *bug* tecnici e implementare nuove funzionalità in modo armonico e coerente con il resto del sistema.

2.4.3.6 Verificatore Il Verificatore[G] è il "garante della qualità" all'interno del team. Il suo compito è assicurarsi che ogni elemento prodotto, dal singolo documento di testo fino al codice finale, rispetti fedelmente i requisiti stabiliti e gli standard prefissati.

Nello specifico, chi ricopre questo ruolo si fa carico di:

- **Revisione documentale:** Leggere e controllare i documenti redatti dagli altri membri del team (tramite le *Pull Request*[G]), assicurandosi che siano corretti, completi e privi di ambiguità.
- **Esecuzione dei collaudi:** Pianificare e condurre le varie fasi di *test* sul software (come i collaudi funzionali, di integrazione e di sistema) per verificare il corretto comportamento delle funzionalità.
- **Segnalazione e tracciamento:** Registrare accuratamente i risultati delle verifiche, segnalando al più presto i difetti o le non conformità agli sviluppatori o agli analisti per permetterne la correzione.
- **Miglioramento continuo:** Individuare eventuali punti critici o inefficienze nel modo in cui il team lavora, proponendo soluzioni pratiche per ottimizzare i processi operativi e di sviluppo.

2.4.4 Tracciamento delle Ore

Il gruppo *aLittleByte* adotta un sistema di rendicontazione oraria basato su un **foglio di calcolo condiviso su Google Fogli**, accessibile a tutti i membri del team.

2.4.4.1 Strumento Le ore vengono registrate all'interno di un foglio Google Fogli condiviso, organizzato per membro e per ruolo ricoperto. Per ciascun task svolto, il membro indica le ore previste e le ore effettivamente impiegate, consentendo il calcolo automatico degli scostamenti e del saldo ore rimanente.

2.4.4.2 Frequenza di aggiornamento Ogni membro del gruppo aggiorna il proprio registro orario **entro la fine di ogni settimana**. Il Responsabile[G] di turno effettua una revisione complessiva del bilancio orario al **termine di ogni Sprint[G]**, verificando che i dati siano completi e coerenti con le attività svolte.

2.4.4.3 Catalogazione delle ore Le ore sono catalogate secondo due dimensioni:

- **Per ruolo:** Responsabile[G], Amministratore[G], Analista[G], Progettista[G], Programmatore[G], Verificatore[G].
- **Per membro:** ciascun componente del gruppo registra separatamente le ore svolte in ogni ruolo, permettendo di tenere traccia del monte ore residuo individuale per ciascuna figura professionale.

2.4.4.4 Monitoraggio Il Responsabile[G] ha il compito di controllare, a fine Sprint[G], che nessun membro abbia esaurito prematuramente le ore assegnate a un determinato ruolo. In caso di criticità, provvede a ridistribuire i task nelle iterazioni successive.

2.4.5 Tracciamento delle Azioni

Al termine di ogni riunione, il membro designato redige il verbale seguendo la struttura definita nella sezione 2.2.1.5. Ogni compito emerso viene immediatamente trasformato in una *Issue*[G] su GitHub[G] e assegnato al membro o ai membri incaricati. Qualora il componente assegnato riscontri difficoltà, può utilizzare la sezione commenti della *Issue*[G] per chiedere chiarimenti al resto del team.

2.4.6 Comunicazione

2.4.6.1 Comunicazioni Interne Le comunicazioni interne avvengono tramite:

- **WhatsApp:** Viene utilizzato per le comunicazioni rapide e coordinamento tra i membri del gruppo.
- **Discussioni GitHub[G]:** Viene utilizzato per discussioni all'interno delle *Issue*[G] e delle *Pull Request*[G], per garantire il tracciamento delle decisioni. Sono utili in quanto permettono di comunicare solo con i membri del gruppo relativi alla discussione.

2.4.6.2 Comunicazioni Esterne Per interagire con l'azienda proponente[G] e con il corpo docente, il gruppo ha definito dei canali di comunicazione specifici, differenziati in base al livello di formalità e all'urgenza della richiesta:

- **E-mail:** Rappresenta il canale ufficiale per le comunicazioni formali, l'invio della documentazione e gli aggiornamenti periodici sullo stato di avanzamento del progetto. Tutte le comunicazioni via posta elettronica vengono gestite tramite l'indirizzo ufficiale del gruppo (`alittlebyte.swe@gmail.com`).
- **Telegram:** Impiegato per scambi di messaggi rapidi, chiarimenti immediati e per il coordinamento logistico direttamente con il referente dell'azienda proponente[G].

2.4.7 Riunioni

2.4.7.1 Riunioni Interne Gli incontri interni del gruppo *aLittleByte* sono pianificati in modo metodico per mantenere un coordinamento efficace e assicurare un avanzamento costante del progetto. Il team ha strutturato i propri momenti di confronto secondo queste modalità:

- **Incontro settimanale:** Di norma, il gruppo si riunisce una volta alla settimana (indicativamente il lunedì). Questo momento di allineamento serve per fare il punto della situazione, organizzare i *task* per i giorni successivi e cercare soluzioni condivise agli ostacoli emersi durante il lavoro individuale.
- **Riunioni straordinarie:** Oltre all'appuntamento fisso, qualora sorgano urgenze, *bug* bloccanti o si debbano prendere decisioni tecniche complesse, i membri interessati possono convocare agilmente degli incontri.
- **Luoghi e piattaforme:** Per comodità organizzativa e per ottimizzare i tempi, le riunioni si svolgono prevalentemente online utilizzando i canali vocali della piattaforma **Discord[G]**.

- **Verbalizzazione:** A garanzia della massima trasparenza operativa, al termine di ogni incontro viene redatto un Verbale Interno[G].

2.4.7.2 Riunioni Esterne I colloqui con l'azienda proponente[G] sono essenziali per garantire il continuo allineamento sugli obiettivi e raccogliere *feedback* sul lavoro svolto. Tutte le sedute si tengono in via telematica sfruttando la piattaforma **Google Meet**. A conclusione di ogni colloquio, il team si occupa di redigere un Verbale Esterno[G] formale che riepiloga puntualmente le tematiche affrontate, le decisioni prese e le future azioni concordate. Tale documento viene infine inoltrato al referente aziendale per un controllo incrociato e, previa conferma di aderenza con quanto discusso, sottoposto alla sua firma per approvazione definitiva.

2.4.8 Processo di Miglioramento Continuo

In conformità con i principi agili e gli standard di qualità adottati, il gruppo aLittleByte applica sistematicamente un ciclo di miglioramento continuo del proprio *Way of Working*[G]. Al termine di ogni iterazione o fase di progetto, il team analizza le misurazioni raccolte nel cruscotto di valutazione all'interno del **Piano di Qualifica**[G].

Qualora emergano metriche sotto la soglia di accettabilità o inefficienze operative, il gruppo elabora specifiche "Iniziative di miglioramento", documentandole nel Piano di Qualifica[G] stesso. Tali iniziative non rimangono semplici osservazioni, ma vengono obbligatoriamente recepite e formalizzate all'interno delle presenti **Norme di Progetto**[G] sotto forma di nuove regole operative e procedurali per gli sprint[G] successivi. Questo meccanismo garantisce che il metodo di lavoro si evolva costantemente, riducendo i rischi, correggendo le anomalie e massimizzando l'efficienza complessiva del team.

3 Qualità del software

Con l'espressione "qualità del software" si indica la totalità delle proprietà di un'entità che concorrono a soddisfare le necessità degli utenti, siano esse dichiarate esplicitamente o sottintese. Tale valutazione si focalizza su due direttrici fondamentali: il valore del prodotto finito e l'efficacia del processo di realizzazione.

L'eccellenza del risultato finale scaturisce invariabilmente dal valore del percorso produttivo intrapreso. Per garantire un software di alto livello, la strategia di valutazione deve essere programmata con cura e non limitata ad una sola verifica finale: tale approccio favorisce una gestione proattiva basata sul monitoraggio costante, superando la logica reattiva volta alla semplice risoluzione delle anomalie.

L'intero processo di valutazione si articola in quattro fasi principali (definizione preventiva delle metriche, misurazione, valutazione dei dati e accettazione finale).

3.1 Qualità del prodotto

Per definire e misurare la qualità intrinseca del software, lo standard di riferimento adottato è la serie di norme ISO/IEC 9126. Questo standard internazionale organizza la qualità in due diverse categorie:

1. **Qualità del Prodotto**, che analizza le seguenti proprietà intrinseche del software:
    - Funzionalità;
    - Affidabilità;
    - Efficienza;
    - Usabilità;
    - Mantenibilità;
    - Portabilità.
2. **Qualità in Uso**, che valuta la capacità del software di permettere agli utenti di raggiungere i propri obiettivi con efficacia, produttività, sicurezza e soddisfazione in specifici contesti d'uso.

Al fine di ottenerne una misurazione oggettiva, sono state stabilite specifiche metriche di prodotto, le quali trovano puntuale rendicontazione e misurazione all'interno del *Piano di Qualifica*^{G}. Tali metriche consentono di analizzare in termini quantitativi elementi determinanti sia del codice sorgente che dell'applicativo finale, tra cui la funzionalità, l'affidabilità, l'efficienza, l'usabilità, la manutenibilità e la portabilità.

3.2 Qualità del processo

Lo standard di riferimento per la qualità del processo è l'ISO/IEC 12207. Questo standard fornisce le linee guida per la gestione della qualità nei processi di sviluppo, includendo la pianificazione^{G}, il controllo e il miglioramento continuo.

Secondo questo standard la qualità del processo si basa su diversi principi fondamentali:

- **Focalizzazione su Scopo e Risultati (Purpose and Outcomes):** Questo rappresenta il pilastro fondamentale. Per garantire la qualità, ogni fase (dall'approvvigionamento alla creazione, fino alla verifica) deve avere finalità ben definite e generare esiti attesi che siano concreti, misurabili e verificabili.
- **Approccio Modulare (Modularity):** La struttura dei processi deve basarsi su componenti caratterizzate da "forte coesione e debole accoppiamento". Ogni singolo modulo deve concentrarsi su un unico traguardo ed essere autonomo, così da limitare le ripercussioni sull'intero ciclo produttivo in caso di variazioni o criticità in un'area specifica.
- **Attribuzione delle Responsabilità (Responsibility):** Seguendo il principio per cui una responsabilità indistinta equivale a nessuna responsabilità, lo standard richiede che ogni attività sia assegnata a un ruolo preciso (come Sviluppatore, Fornitore, Acquisitore o Verificatore^{G}), eliminando incertezze all'interno dell'organizzazione.
- **Personalizzazione dei Processi (Tailoring):**Data la diversità tra i vari progetti software, lo standard prevede il principio del Tailoring. Le aziende devono adattare i processi alle proprie esigenze, selezionando esclusivamente le procedure pertinenti

alla complessità, al rischio[G] e alle risorse disponibili, evitando carichi burocratici superflui.

- **Evoluzione Continua e Tracciabilità:**È essenziale che ogni intervento, anomalia o variazione sia riconducibile al requisito di partenza. Il sistema non è statico, poiché integra cicli di audit, revisione e gestione delle problematiche. Ciò permette al gruppo di lavoro di ottimizzare costantemente le proprie metodologie, avvalendosi spesso di modelli quali CMMI o SPICE per attestare il livello di maturità raggiunto.

3.2.0.1 Applicazione del Tailoring Lo standard ISO/IEC 12207:1997 viene adattato al contesto di un team accademico che sviluppa un MVP[G] stand-alone con moduli AI[G]. Sono mantenuti i processi di **Fornitura** e **Sviluppo** (adattati con Agile[G]/SCRUM), mentre Acquisizione, Operazione e Manutenzione sono esclusi. La documentazione è ridotta ai soli artefatti richiesti dalla baseline; la gestione della configurazione è delegata a GitHub[G]. L'efficacia del tailoring è monitorata tramite la metrica **MPC-QMS**: un valore sotto soglia segnala la necessità di rivedere le scelte nell'iterazione successiva.

In questo progetto, tale obiettivo viene perseguito attraverso una precisa delineazione di metodologie e strumenti operativi. Il controllo costante dell'avanzamento è garantito da sistemi di tracciamento dei problemi (*issue*[G] *tracking*) e da incontri periodici formalizzati, supportati da un'analisi rigorosa dei requisiti e da verifiche regolari sugli standard qualitativi dei risultati ottenuti.

Al fine di assicurare un monitoraggio imparziale, il gruppo ha stabilito specifiche metriche di processo, dettagliate all'interno del *Piano di Qualifica*[G]. Tali indicatori permettono di misurare l'efficienza del ciclo di sviluppo, focalizzandosi su variabili critiche quali la puntualità nelle consegne, l'aderenza al budget stabilito, la frequenza di anomalie rilevate, l'estensione dei test e la conformità complessiva ai criteri prefissati.

4 Metriche

Questa sezione definisce in modo formale tutte le metriche adottate dal gruppo per quantificare la qualità del processo di sviluppo e del prodotto software finale. I valori di soglia (accettabili e ottimali) e i risultati effettivi delle misurazioni sono documentati nel *Piano di Qualifica*[G], che contiene il cruscotto di valutazione con l'andamento storico delle misurazioni.

Ogni metrica è descritta in termini di:

- **Identificativo:** codice univoco della metrica;
- **Nome:** denominazione della metrica;
- **Descrizione:** definizione operativa dello scopo della metrica;
- **Formula:** espressione matematica per il calcolo del valore;
- **Unità di misura:** unità in cui il valore è espresso.

Le metriche di processo sono identificate dalla sigla **MPC** (Metrica di Processo e Controllo), mentre le metriche di prodotto dalla sigla **MPD** (Metrica di Prodotto).

4.1 Metriche per la qualità di processo

Queste metriche misurano l'efficacia, l'efficienza e il controllo delle attività necessarie per sviluppare il prodotto. Il riferimento metodologico adottato è lo standard ISO/IEC 12207, adattato al Way of Working[G] del gruppo. Le metriche non valutano il prodotto finito, ma il modo in cui esso viene realizzato.

4.1.1 Processi primari — Fornitura

4.1.1.1 MPC-EV — Earned Value

- **Descrizione:** Valore del lavoro effettivamente completato, misurato come la quota di valore pianificato corrispondente alle attività portate a termine nello sprint[G].
- **Formula:**
$$\text{EV} = \text{PV} \times \frac{\text{Ore Completate}}{\text{Ore Pianificate}}$$
- **Unità di misura:** Euro

4.1.1.2 MPC-PV — Planned Value

- **Descrizione:** Valore economico del lavoro pianificato per lo sprint[G], estratto dai preventivi del Piano di Progetto[G]. Rappresenta il costo previsto per le attività programmate.
- **Formula:**
$$\text{PV} = \sum_{i=1}^{n} (\text{Ore Previste}_i \times \text{Tariffa Oraria}_i)$$
dove n è il numero di task pianificate nello sprint[G] e la tariffa oraria corrisponde al ruolo assegnato alla task.
- **Unità di misura:** Euro

4.1.1.3 MPC-AC — Actual Cost

- **Descrizione:** Costo effettivamente sostenuto per le ore lavorate nello sprint[G], ricavato dai consuntivi del Piano di Progetto[G].
- **Formula:**
$$\text{AC} = \sum_{i=1}^{n} (\text{Ore Effettive}_i \times \text{Tariffa Oraria}_i)$$
dove n è il numero di task completate nello sprint[G] e la tariffa oraria corrisponde al ruolo assegnato alla task.
- **Unità di misura:** Euro

4.1.1.4 MPC-CPI — Cost Performance Index

- **Descrizione:** Misura l'efficienza dei costi del progetto. Un valore inferiore a 1 indica che il progetto sta spendendo più del valore prodotto; un valore pari a 1 indica perfetta efficienza di costo.
- **Formula:**
$$\mathrm{CPI} = \frac{\mathrm{EV}_{\mathrm{cum}}}{\mathrm{AC}_{\mathrm{cum}}}$$
dove i valori cumulativi sommano tutti gli sprint[G] dall'inizio del progetto.
- **Interpretazione:**
  - CPI > 1: il progetto produce valore a un costo inferiore al previsto
  - CPI $= 1$: il progetto è in linea con il budget
  - CPI < 1: il progetto sta spendendo più del valore prodotto
- **Unità di misura:** adimensionale

4.1.1.5 MPC-SPI — Schedule Performance Index

- **Descrizione:** Misura l'efficienza della pianificazione[G] temporale. Un valore inferiore a 1 indica ritardo rispetto al piano; un valore pari a 1 indica perfetto rispetto delle scadenze.
- **Formula:**
$$\mathrm{SPI} = \frac{\mathrm{EV}_{\mathrm{cum}}}{\mathrm{PV}_{\mathrm{cum}}}$$
- **Interpretazione:**
  - SPI > 1: il progetto è in anticipo rispetto al piano
  - SPI $= 1$: il progetto è in linea con il piano
  - SPI < 1: il progetto è in ritardo rispetto al piano
- **Unità di misura:** adimensionale

4.1.1.6 MPC-EAC — Estimated At Completion

- **Descrizione:** Stima del costo totale finale del progetto proiettando l'efficienza di costo corrente (CPI) sull'intero budget rimanente.
- **Formula:**
$$\mathrm{EAC} = \frac{\mathrm{BAC}}{\mathrm{CPI}}$$
- **Interpretazione:**
  - EAC $<$ BAC: il progetto si concluderà sotto budget
  - EAC $=$ BAC: il progetto rispetterà il budget
  - EAC $>$ BAC: il progetto supererà il budget
- **Unità di misura:** Euro

4.1.1.7 MPC-ETC — Estimate To Complete

- **Descrizione:** Stima del costo rimanente necessario per completare il progetto a partire dallo sprint[G] corrente, ottenuta sottraendo il costo cumulativo effettivo dalla stima a completamento.
- **Formula:**
$$\text{ETC} = \text{EAC} - \text{AC}_{\text{cum}}$$
dove AC_{cum} è la somma dei costi effettivi sostenuti dall'inizio del progetto fino allo sprint[G] corrente.
- **Unità di misura:** Euro

4.1.1.8 MPC-TEAC — Time Estimate At Completion

- **Descrizione:** Stima della durata totale del progetto a partire dalla performance di avanzamento misurata dallo SPI. Se SPI = 1, il progetto terminerà esattamente nei tempi pianificati.
- **Formula:**
$$\text{TEAC} = \frac{\text{Durata pianificata}}{\text{SPI}}$$
- **Unità di misura:** settimane

4.1.2 Processi primari — Sviluppo

4.1.2.1 MPC-RSI — Requirements Stability Index

- **Descrizione:** Misura la stabilità dei requisiti nel tempo, confrontando il numero di requisiti modificati, eliminati o aggiunti rispetto alla baseline iniziale dell'Analisi dei Requisiti[G]. Valori prossimi al 100% indicano un documento dei requisiti stabile e maturo.
- **Formula:**
$$\text{RSI} = \left(1 - \frac{N_{\text{modificati}} + N_{\text{eliminati}} + N_{\text{aggiunti}}}{N_{\text{totali iniziali}}}\right) \times 100\%$$
- **Unità di misura:** percentuale (%)

4.1.3 Processi di supporto — Documentazione

4.1.3.1 MPC-IG — Indice di Gulpease

- **Descrizione:** Indice di leggibilità calibrato per la lingua italiana. Valuta la complessità del testo in base alla lunghezza delle parole e delle frasi rispetto al numero totale di parole. Valori più alti indicano maggiore leggibilità. Il calcolo viene effettuato tramite estrazione del testo con `detex`[G] e script Python.
- **Formula:**
$$G = 89 - \frac{10 \cdot L}{P} + \frac{300 \cdot F}{P}$$
dove L è il numero di lettere, P il numero di parole e F il numero di frasi.
- **Unità di misura:** adimensionale (scala 0–100)

4.1.3.2 MPC-CO — Correttezza Ortografica

- **Descrizione:** Numero di errori grammaticali o di battitura rilevati nei documenti tramite strumenti automatici di controllo ortografico. La verifica viene condotta tramite estrazione del testo con `detex`[G] e analisi automatica. Il valore accettabile e ottimale è zero errori.
- **Formula:**

$$\text{CO} = \text{Numero di Errori Ortografici Rilevati}$$

- **Unità di misura:** numero intero (conteggio assoluto)

4.1.4 Processi di supporto — Verifica

4.1.4.1 MPC-CC — Code Coverage

- **Descrizione:** Percentuale di righe di codice sorgente effettivamente eseguite durante i test automatici. Misura il grado di copertura complessivo della suite di test. La metrica è applicabile a partire dalla fase PB[G], quando sarà disponibile il codice sorgente.
- **Formula:**

$$\text{CC} = \frac{\text{Linee di Codice Eseguite dai Test}}{\text{Linee di Codice Totali}} \times 100$$

- **Unità di misura:** percentuale (%)

4.1.4.2 MPC-TSR — Test Success Rate

- **Descrizione:** Percentuale di test automatici che hanno restituito un esito positivo sul totale dei test eseguiti. Un valore del 100% indica l'assenza di regressioni nella suite corrente.
- **Formula:**

$$\text{TSR} = \frac{\text{Test Superati}}{\text{Test Eseguiti}} \times 100$$

- **Unità di misura:** percentuale (%)

4.1.5 Processi di supporto — Gestione della qualità

4.1.5.1 MPC-QMS — Quality Metrics Satisfied

- **Descrizione:** Percentuale di metriche di qualità (di processo e di prodotto) che rientrano nel range accettabile definito nel Piano di Qualifica[G]. Rappresenta un indicatore sintetico della salute complessiva del progetto al termine di ciascuno sprint[G].
- **Formula:**

$$\text{QMS} = \frac{\text{Metriche soddisfatte}}{\text{Metriche misurate}} \times 100\%$$

- **Unità di misura:** percentuale (%)

4.1.6 Processi organizzativi — Gestione dei processi

4.1.6.1 MPC-TE — Time Efficiency

- **Descrizione:** Rapporto tra le ore dedicate ad attività produttive effettivamente rendicontabili e le ore totali disponibili nello sprint[G]. Indica quanta parte del tempo disponibile si traduce direttamente in avanzamento del progetto.
- **Formula:**
$$\text{TE} = \frac{\text{Ore produttive}}{\text{Ore totali disponibili}} \times 100\%$$
- **Unità di misura:** percentuale (%)

4.1.6.2 MPC-PRI — Percentuale Rischi Inattesi

- **Descrizione:** Percentuale di rischi effettivamente manifestatisi che non erano stati identificati nel registro dei rischi del Piano di Progetto[G]. Valori bassi indicano una pianificazione[G] del rischio[G] efficace e un registro dei rischi completo.
- **Formula:**
$$\text{PRI} = \frac{N_{\text{rischi inattesi}}}{N_{\text{rischi totali manifestati}}} \times 100\%$$
- **Unità di misura:** percentuale (%)

4.1.6.3 MPC-LE — Labor Efficiency

- **Descrizione:** Rapporto tra le ore dedicate ad attività a valore aggiunto (che producono direttamente un output di progetto) e il totale delle ore lavorate. Misura l'efficienza nell'impiego del lavoro del team, escludendo le attività non direttamente produttive.
- **Formula:**
$$\text{LE} = \frac{\text{Ore a valore aggiunto}}{\text{Ore totali lavorate}} \times 100\%$$
- **Unità di misura:** percentuale (%)

4.2 Metriche per la qualità di prodotto

Queste metriche misurano le proprietà interne ed esterne del software finale per garantire che il sistema rispetti i requisiti. Lo standard di riferimento è ISO/IEC 9126, adottato come base per la classificazione delle caratteristiche di qualità del prodotto.

4.2.1 Funzionalità

4.2.1.1 MPD01 — Requisiti Obbligatori Soddisfatti

- **Descrizione:** Percentuale di requisiti obbligatori correttamente implementati e verificati rispetto al totale dei requisiti obbligatori rilevati nell'Analisi dei Requisiti[G]. Un requisito è considerato soddisfatto quando il relativo test di sistema[G] risulta superato.
- **Formula:**
$$\text{MPD01} = \frac{\text{Requisiti Obbligatori Soddisfatti}}{\text{Requisiti Obbligatori Totali}} \times 100$$
- **Unità di misura:** percentuale (%)

4.2.1.2 MPD02 — Requisiti Opzionali Soddisfatti

- **Descrizione:** Percentuale di requisiti opzionali[G] correttamente implementati e verificati rispetto al totale dei requisiti opzionali[G] rilevati. Un requisito è considerato soddisfatto quando il relativo test di sistema[G] risulta superato.
- **Formula:**
$$\text{MPD02} = \frac{\text{Requisiti Opzionali}^{\text{G}}\text{ Soddisfatti}}{\text{Requisiti Opzionali}^{\text{G}}\text{ Totali}} \times 100$$
- **Unità di misura:** percentuale (%)

4.2.1.3 MPD03 — AI Acceptance Rate

- **Descrizione:** Percentuale di generazioni AI[G] valutate dagli utenti con un rating[G] pari o superiore a 3 su 5 rispetto al totale delle generazioni valutate. Misura la qualità percepita delle risposte prodotte dai moduli AI[G] Assistant e AI[G] Co-Pilot del sistema.
- **Formula:**
$$\text{MPD03} = \frac{\text{Generazioni con rating}^{\text{G}} \geq 3/5}{\text{Generazioni totali valutate}} \times 100$$
- **Unità di misura:** percentuale (%)

4.2.2 Affidabilità

4.2.2.1 MPD04 — Branch Coverage

- **Descrizione:** Percentuale di rami decisionali (ad esempio i rami *true* e *false* di un costrutto `if`) eseguiti durante i test automatici. Garantisce che tutte le possibili direzioni del flusso logico siano state verificate dalla suite di test.
- **Formula:**
$$\text{MPD04} = \frac{\text{Rami Decisionali Eseguiti}}{\text{Rami Decisionali Totali}} \times 100$$
- **Unità di misura:** percentuale (%)

4.2.2.2 MPD05 — Defect Density

- **Descrizione:** Densità di difetti nel codice sorgente, calcolata come numero di difetti rilevati per ogni KLOC^{G} (migliaia di righe di codice). Valori bassi indicano maggiore qualità del codice prodotto e minore frequenza di anomalie.
- **Formula:**
$$\text{MPD05} = \frac{\text{Numero di Difetti Rilevati}}{\text{KLOC}^{G}}$$
dove KLOC^{G} = Righe di Codice Totali/1000.
- **Unità di misura:** difetti per KLOC^{G}

4.2.3 Efficienza

4.2.3.1 MPD06 — Latenza Interfaccia Utente

- **Descrizione:** Tempo di risposta del sistema a un'interazione dell'utente sull'interfaccia, misurato come intervallo tra l'invio della richiesta e la ricezione della risposta completa per le operazioni generali dell'applicazione (escluse le chiamate ai moduli AI^{G}).
- **Formula:**
$$\text{MPD06} = T_{\text{risposta}} - T_{\text{richiesta}}$$
- **Unità di misura:** secondi (s)

4.2.3.2 MPD07 — Latenza Generazione Testo AI Assistant

- **Descrizione:** Tempo intercorso tra l'invio del prompt^{G} al modulo AI Post Generator^{G} e la ricezione del testo della bozza^{G} completa generata dal servizio AI^{G}.
- **Formula:**
$$\text{MPD07} = T_{\text{risposta AI}^{G}} - T_{\text{invio prompt}^{G}}$$
- **Unità di misura:** secondi (s)

4.2.3.3 MPD08 — Latenza Generazione Immagine AI Assistant

- **Descrizione:** Tempo intercorso tra la richiesta di generazione dell'immagine di copertina al modulo AI^{G} Assistant e la ricezione dell'immagine generata e pronta per la visualizzazione.
- **Formula:**
$$\text{MPD08} = T_{\text{ricezione immagine}} - T_{\text{richiesta immagine}}$$
- **Unità di misura:** secondi (s)

4.2.3.4 MPD09 — Latenza Analisi Documentale AI Co-Pilot

- **Descrizione:** Tempo intercorso tra il caricamento di un documento e la restituzione dei metadati[G] classificati dal modulo AI Analyst[G] (classificazione tipologia, estrazione destinatario, livello di confidenza[G]).
- **Formula:**

$$\text{MPD09} = T_{\text{risultato analisi}} - T_{\text{caricamento documento}}$$

- **Unità di misura:** secondi (s)

4.2.4 Usabilità

4.2.4.1 MPD10 — Profondità Massima di Navigazione

- **Descrizione:** Numero massimo di livelli gerarchici che un utente deve attraversare per raggiungere qualsiasi funzionalità del sistema a partire dalla schermata principale. Valori bassi indicano un'interfaccia piatta e facilmente accessibile.
- **Formula:**

$$\text{MPD10} = \max\{\text{Livelli di navigazione necessari per raggiungere la funzionalità } f,\ \forall f\}$$

- **Unità di misura:** livelli di navigazione

4.2.4.2 MPD11 — Learning Time

- **Descrizione:** Tempo medio necessario a un utente privo di esperienza pregressa con il sistema per apprendere le funzionalità principali in modo autonomo, misurato tramite sessioni di user testing. La verifica avviene confrontando il tempo con la soglia accettabile di 30 minuti.
- **Formula:**

$$\text{MPD11} = \frac{\sum_{i=1}^{n} T_{\text{apprendimento}_i}}{n}$$

dove n è il numero di utenti coinvolti nella sessione di test e $T_{\text{apprendimento}_i}$ è il tempo impiegato dall'utente i per completare le funzionalità principali senza assistenza.

- **Unità di misura:** minuti

4.2.5 Mantenibilità

4.2.5.1 MPD12 — Code Smell

- **Descrizione:** Numero di code smell rilevati nel codice sorgente tramite analisi statica[G]. I code smell sono indicatori di progettazione debole (metodi troppo lunghi, classi troppo grandi, duplicazione di codice) che, pur non essendo errori bloccanti, rendono il sistema fragile e difficile da manutenere nel tempo.
- **Formula:**

$$\text{MPD12} = \text{Numero di Code Smell Rilevati}$$

- **Unità di misura:** numero intero (conteggio assoluto)

4.2.5.2 MPD13 — Coefficient of Coupling

- **Descrizione:** Misura il grado di interdipendenza tra i moduli del software. Un accoppiamento elevato implica che una modifica in un modulo rischi di propagarsi ad altri (effetto a catena), riducendo la manutenibilità del sistema.
- **Formula:**

$$\text{MPD13} = \frac{\text{Dipendenze Effettive tra Moduli}}{n \times (n-1)}$$

  dove n è il numero totale di moduli del sistema e $n \times (n-1)$ rappresenta il numero massimo di dipendenze direzionali possibili. Una dipendenza sussiste quando un modulo importa, invoca o referenzia direttamente un altro modulo.
- **Interpretazione:**
  - MPD13 = 0: nessun accoppiamento (moduli completamente indipendenti)
  - MPD13 = 1: accoppiamento massimo (ogni modulo dipende da tutti gli altri)
- **Unità di misura:** adimensionale (scala 0–1)

4.2.5.3 MPD14 — Cyclomatic Complexity

- **Descrizione:** Misura la complessità logica del codice contando il numero di percorsi linearmente indipendenti attraverso il flusso di controllo (costrutti `if`, `switch`, cicli). Valori elevati indicano codice difficile da testare e comprendere. Il valore riportato nel cruscotto è riferito al singolo metodo.
- **Formula:**

$$M = E - N + 2P$$

  dove E è il numero di archi nel grafo di controllo del flusso, N il numero di nodi (istruzioni) e P il numero di componenti connesse del grafo (solitamente $P = 1$ per una singola funzione).
- **Unità di misura:** numero intero (adimensionale)

4.2.6 Portabilità

4.2.6.1 MPD15 — Compatibilità Cross-Browser

- **Descrizione:** Percentuale di browser target (principali browser moderni evergreen nelle ultime due versioni stabili) sui quali il sistema funziona correttamente senza degradazione delle funzionalità o dell'interfaccia utente.
- **Formula:**

$$\text{MPD15} = \frac{\text{Browser Supportati correttamente}}{\text{Browser Target Totali}} \times 100$$

- **Unità di misura:** percentuale (%)