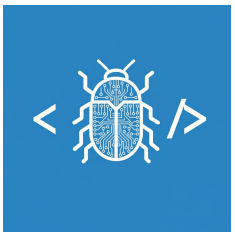

BugBusters

Email: bugbusters.unipd@gmail.com

Gruppo: 4

Università degli Studi di Padova

Laurea in Informatica

Corso: Ingegneria del Software

Anno Accademico: 2025/2026

Norme di Progetto$_G$

Versione 2.0.0

Stato	Approvato
Redattori$_G$	Alberto Autiero, Marco Favero, Alberto Pignat, Marco Piro, Linor Sadé, Leonardo Salviato, Luca Slongo
Verificatori$_G$	Alberto Pignat, Linor Sadé, Luca Slongo
Destinatari	BugBusters, Prof. Tullio Vardanega, Prof. Riccardo Cardin

Abstract

Documento contenente le Norme di Progetto$_G$ adottate dal team BugBusters per lo sviluppo del progetto$_G$ Nexum proposto dall'azienda Eggon. Il documento include metodologie di lavoro, standard di codifica, processi di sviluppo e gestione del progetto$_G$.

Registro delle modifiche

Versione	Data	Descrizione	Redatto	Verificato	Approvato
2.0.0	01/03/2026	Approvato per PB$_G$	-	-	Marco Favero
1.4.0	28/02/2026	Verifica$_G$ finale	-	Alberto Pignat	-
1.3.1	28/02/2026	Aggiunta sezione Formazione	Leonardo Salviato	-	-
1.3.0	26/02/2026	Verificata sezione accertamento e sezione rischi	-	Alberto Pignat	-
1.2.1	25/02/2026	Aggiunta sezione progettazione e codifica	Leonardo Salviato	-	-
1.2.0	25/02/2026	Verificata sezione progettazione e codifica	-	Linor Sadè	-
1.1.1	24/02/2026	Aggiunta sezione progettazione e codifica	Leonardo Salviato	-	-
1.1.0	23/02/2026	Verificata sezione miglioramento continuo	-	Linor Sadè	-
1.0.1	23/02/2026	Aggiunta sezione miglioramento continuo	Leonardo Salviato	-	-
1.0.0	30/01/2026	Approvazione$_G$ documento	-		Luca Slongo
0.2.0	30/01/2026	Verifica$_G$ documento	-	Alberto Pignat	-
0.1.7	29/01/2026	Aggiunti ai termini presenti nel Glossario$_G$ la G	Luca Slongo	-	-
0.1.6	19/1/2026	Terminate metriche finali	Linor Sadè	-	-
0.1.5	18/1/2026	Aggiornamento sezione qualità$_G$, aggiunti dettagli su metriche di processo	Linor Sadè	-	-
0.1.4	13/1/2026	continuo sezione qualità$_G$, aggiunti: standard e modelli, principi di qualità$_G$ del processo, metriche di processo	Linor Sadè	-	-
0.1.3	12/1/2026	Inizio sezione qualità$_G$, modificato inizio precedente	Linor Sadè	-	-
0.1.2	4/1/2026	Aggiunte parti mancanti relative al Piano di Qualifica$_G$. Ulteriori correzioni al contenuto precedente.	Linor Sadè	-	-
0.1.1	3/1/2026	Correzioni minime nel documento e aggiunte alcune parti mancanti	Linor Sadè	-	-
0.1.0	30/12/2025	Verificato	-	Luca Slongo	-

0.0.10	22/12/2025	Aggiunto Piano di Progetto$_G$ e di Qualifica	Marco Piro	-	-
0.0.9	18/12/2025	Eliminato i Processi di acquisizione ed aggiunto le lettere di Candidatura$_G$ e Presentazione	Marco Piro	-	-
0.0.8	04/12/2025	Conclusione momentanea della sezione relativa ai processi di supporto	Alberto Pignat	-	-
0.0.7	02/12/2025	Avanzamento processi di supporto: Configurazioni e Qualifica	Alberto Pignat	-	-
0.0.6	28/11/2025	Iniziati processi di supporto: documentazione	Marco Favero	-	-
0.0.5	27/11/2025	Correzione strutturale e sistemazione lavoro svolto	Marco Favero	-	-
0.0.4	20/11/2025	Finita la parte di Documentazione prodotta in processi di fornitura	Marco Favero	-	-
0.0.3	19/11/2025	Avanzamento, parte di documentazione prodotta	Marco Favero	-	-
0.0.2	16/11/2025	Introduzione, processi di acquisizione e iniziati processi di fornitura	Marco Favero	-	-
0.0.1	04/11/2025	Prima stesura della struttura del documento dopo l'aggiudicazione dell'appalto per il capitolato$_G$ C5 Nexum dell'azienda Eggon	Alberto Autiero	-	-

Indice

1 Introduzione

1.1 Scopo del documento

Questo documento definisce le Norme di Progetto$_G$ adottate dal team BugBusters per lo sviluppo del progetto$_G$ Nexum, proposto dall'azienda Eggon. Le Norme di Progetto$_G$ includono metodologie di lavoro, standard di codifica, processi di sviluppo e gestione del progetto$_G$, al fine di garantire un approccio strutturato e coerente durante l'intero ciclo di vita del progetto$_G$.

Per strutturare il nostro way of working$_G$, faremo riferimento alle best practice$_G$ suggerite dallo standard ISO/IEC 12207:1995 adattandole alle esigenze specifiche del nostro team e del progetto$_G$ Nexum. In particolare lo standard identifica tre tipologie di processi:

- Processi primari: processi direttamente coinvolti nella creazione del prodotto$_G$ software
- Processi di supporto: processi che supportano i processi primari
- Processi organizzativi: processi che gestiscono e coordinano le attività del team

La combinazione di questi processi ci permetterà di gestire in modo efficace lo sviluppo del progetto$_G$ Nexum. La stesura di questo documento mira a fornire una guida chiara e condivisa per tutti i membri del team, assicurando che le attività di sviluppo siano svolte in modo efficiente e conforme agli standard di qualità$_G$ previsti. Il documento è redatto in maniera incrementale: verrà aggiornato e ampliato progressivamente durante lo sviluppo del progetto$_G$ per riflettere decisioni, modifiche e miglioramenti adottati dal team.

1.2 Scopo del prodotto$_G$

Nexum è una piattaforma con un modulo$_G$ dedicato alle comunicazioni interne, alla raccolta di feedback e alla timbratura digitale. Include una messaggistica top-down con tracciamento delle letture, un builder per survey con logiche di ramificazione e dashboard$_G$ in tempo reale; inoltre un sistema di timbratura via badge o dispositivi mobili con regole automatiche per il controllo e l'aggregazione delle ore. Le anagrafiche centralizzate, con ruoli e permessi, permettono una gestione granulare degli accessi e l'integrazione dei moduli, garantendo un flusso informativo coerente, tracciabile e adattabile alle esigenze operative. Il nostro progetto$_G$ mira alla creazione di un'applicazione stand-alone$_G$ (rispetto all'applicazione Nexum già esistente) che implementa un modulo$_G$ di AI$_G$ assistant generativo per la scrittura di comunicazioni e l'AI Co-Pilot$_G$ per i Cdl$_G$. In particolare quest'ultimo deve essere in grado di riconoscere i documenti caricati dagli utenti, estrarne le informazioni rilevanti, capire la tipologia, destinatari e consegnarli in modo massivo.
Per ulteriori informazioni sul progetto$_G$ si rimanda all'Analisi dei Requisiti$_G$. Il nostro obiettivo è realizzare questo progetto$_G$ entro il 21 marzo 2026 con un budget di 12.790 euro.

1.3 Glossario$_G$

Il glossario$_G$ raccoglie e definisce i termini, gli acronimi e le abbreviazioni impiegati nel documento e nel progetto$_G$ Nexum. L'obiettivo è fornire definizioni univoche per ridurre ambiguità, garantire coerenza terminologica tra i membri del team e facilitare l'onboarding di nuovi partecipanti.
Per i termini tecnici e specifici utilizzati in questo documento, si fa riferimento al glossario$_G$ disponibile al seguente link. Per maggiore usabilità e facilità di consultazione, il glossario$_G$ è accessibile anche aprendo dal nostro sito web i vari documenti con il viewer pdf da noi sviluppato.

1.3.1 Riferimenti normativi

- **Capitolato$_G$ d'appalto C5: Nexum - Piattaforma di consulenza e documentazione previdenziale**
  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C5.pdf

1.3.2 Riferimenti informativi

- **Glossario$_G$:**
  https://bugbustersunipd.github.io/DocumentazioneSWE/PB/GLOSSARIO/Glossario.pdf

2 Processi Primari

L'obiettivo principale di BugBusters è la realizzazione di un prodotto$_G$ software di alta qualità$_G$ che soddisfi le esigenze del cliente e degli utenti finali. Per raggiungere questo obiettivo, è indispensabile basarsi su un modello di riferimento che definisca processi chiari da seguire.
L'adozione di un simile framework metodologico, che indirizza ad esempio le fasi di acquisizione e di costruzione del software, è ciò che permette di passare da un semplice programma funzionante a un prodotto$_G$ di valore e di lunga durata. Basandosi dunque sullo standard ISO/IEC 12207:1995, il team BugBusters ha deciso di adottare i seguenti processi primari:

- Processi di fornitura
- Processi di sviluppo

2.1 Processi di fornitura

2.1.1 Scopo

Lo scopo del processo di fornitura è definire e regolamentare l'insieme delle attività preliminari necessarie ad avviare il progetto$_G$ in modo controllato, condiviso e verificabile. In particolare, tale processo ha l'obiettivo di chiarire i requisiti$_G$ richiesti dalla proponente$_G$, identificare vincoli e risorse, pianificare le modalità operative, stabilire gli strumenti di lavoro e formalizzare la documentazione necessaria, così da garantire una corretta base metodologica e contrattuale per le successive fasi di sviluppo.

2.1.2 Attività

La fornitura prevede varie attività, in particolare:

- **Analisi del capitolato$_G$ proposto**: raccolta di informazioni, requisiti$_G$, tecnologie e vincoli presenti nel capitolato$_G$ d'appalto. In questa fase si pongono domande alla proponente$_G$ per chiarire eventuali dubbi o ambiguità, processo utile alla scelta finale.
- **Stima delle risorse**: definizione delle tempistiche, dei costi totali e delle risorse necessarie per la realizzazione del progetto$_G$.
- **Analisi dei Requisiti$_G$ e contrattazione con la proponente$_G$**: identificazione e documentazione dei requisiti$_G$ funzionali e non funzionali del software, analisi delle aspettative del proponente$_G$ e negoziazione di eventuali modifiche o aggiunte. Definizione del Minimum Viable Product (MVP)$_G$.
- **Pianificazione del progetto$_G$**: suddivisione del progetto$_G$ in fasi, definizione delle milestone$_G$ aggiuntive (oltre a RTB$_G$ e PB$_G$), assegnazione dei compiti ai membri del team e pianificazione delle attività. Individuazione degli strumenti di lavoro e delle metodologie da adottare; definizione del Proof of Concept (PoC)$_G$.
- **Documentazione**: redazione di tutti i documenti necessari per formalizzare la fornitura, come il Piano di Progetto$_G$, l'Analisi dei Requisiti$_G$, Norme di Progetto$_G$ e altri documenti di supporto. La documentazione prodotta sarà utilizzata sia come strumento di lavoro interno al team sia come riferimento e strumento di controllo da parte della proponente$_G$.
- **Comunicazione con la proponente$_G$**: stabilire canali di comunicazione efficaci per garantire un flusso informativo continuo e trasparente. Prevedere incontri regolari per aggiornamenti sullo stato del progetto$_G$, discussione di eventuali problemi e raccolta di feedback.

- **Revisione e approvazione$_G$**: sottoporre tutta la documentazione prodotta alla revisione e approvazione$_G$ della proponente$_G$ e del committente$_G$, assicurando l'allineamento sugli obiettivi e le aspettative prima di procedere con le fasi successive del progetto$_G$.
- **Consegna e chiusura della fase di fornitura**: consegna di quanto prodotto$_G$ durante la fase di fornitura al proponente$_G$, garantendo che tutti i documenti siano completi e corretti. Completata la consegna, si procede alla chiusura formale della fase.

2.1.3 Strumenti di Supporto

Per supportare le attività di fornitura, il team BugBusters usufruisce di vari strumenti:

- **GitHub$_G$**: Per la documentazione collaborativa, il versionamento$_G$ dei documenti, la produzione asincrona, il sistema di ticketing e delle project board.
- **GitLab**: Per il codice già esistente fornito dalla proponente$_G$, utile all'implementazione$_G$ di un prodotto$_G$ che possa essere facilmente integrabile alla piattaforma già esistente.
- **Discord$_G$**: Per le riunioni di team.
- **WhatsApp**: Per comunicazioni rapide e aggiornamenti interni al gruppo.
- **Google Calendar**: Per la pianificazione delle riunioni e delle scadenze.
- **Telegram e Gmail**: Per la comunicazione con la proponente$_G$ e per inviare documenti ufficiali.

2.1.4 Comunicazione e Organizzazione

Le comunicazioni con la proponente$_G$ avvengono solitamente a cadenza bisettimanale mercoledì alle 15:00, salvo imprevisti o esigenze particolari. I verbali relativi agli incontri con la proponente$_G$ vengono condivisi e archiviati nel repository$_G$ della documentazione per garantire trasparenza e reperibilità. Tutte le decisioni rilevanti emerse negli incontri con la proponente$_G$ devono essere formalmente riportate e documentate con il sistema di ticketing su GitHub$_G$ e nella documentazione ufficiale. Eventuali problemi critici o dubbi vengono comunicati immediatamente sui canali concordati con la proponente$_G$, in modo da garantirne la tempestività nelle azioni correttive.

2.1.5 Documentazione Prodotta

Durante la fase di fornitura, il team BugBusters produce e mantiene aggiornata la seguente documentazione:

- Glossario$_G$
- Analisi dei Requisiti$_G$
- Piano di Progetto$_G$
- Piano di Qualifica$_G$
- Verbali$_G$
- Lettera di candidatura$_G$
- Specifica tecnica$_G$

- Manuale utente$_G$
- Valutazione dei capitolati
- Norme di Progetto$_G$

2.1.5.1 Glossario$_G$

Per facilitare la lettura e la comprensione dei documenti di progetto$_G$, viene redatto un glossario$_G$ che raccoglie e definisce i termini tecnici, gli acronimi e le abbreviazioni utilizzate. Questo strumento è fondamentale per garantire una comunicazione chiara e univoca tra tutti i membri del team, con la proponente$_G$, docenti (il committente$_G$) e con i lettori esterni. Per una consultazione rapida durante la visualizzazione dei documenti, il glossario$_G$ è accessibile anche tramite il viewer PDF sviluppato dal team che si trova nella repository$_G$ web ufficiale della documentazione accessibile al seguente link.

Campo	Dettaglio
Redattore$_G$	Amministratore$_G$
Destinatari	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno ed Esterno

2.1.5.2 Dichiarazione degli impegni

La Dichiarazione degli Impegni definisce i ruoli, il monte ore previsto, la data di consegna prevista e il costo orario di ciascun membro del team BugBusters per la realizzazione del progetto$_G$ Nexum, impegnandosi a rispettare tali condizioni durante l'intero ciclo di vita del progetto$_G$.

Campo	Dettaglio
Redattore$_G$	Responsabile$_G$
Destinatari	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Esterno

2.1.5.3 Valutazione dei capitolati

Con questo documento si intende valutare i vari capitolati d'appalto proposti per il progetto$_G$ di Ingegneria del Software, analizzandone punti di forza, debolezze e opportunità in modo da scegliere il capitolato$_G$ più adatto alle competenze e agli interessi del team BugBusters. Per ogni capitolato$_G$ vengono esaminati vari aspetti:

- Descrizione breve
- Caratteristiche funzionali
- Tecnologie proposte
- Chiarimenti e colloqui con l'azienda
- Interesse del team
- Punti di forza e debolezza

Campo	Dettaglio
Redattore$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin
Uso	Esterno

2.1.5.4 Analisi dei Requisiti$_G$

Il documento di Analisi dei Requisiti$_G$ descrive in dettaglio le funzionalità$_G$ e i vincoli. Questo documento fornisce:

- la definizione del contesto, degli stakeholder$_G$ e degli attori coinvolti;
- l'elenco dei requisiti$_G$ funzionali$_G$ e non funzionali$_G$, classificati e dotati di codifica univoca e criteri di accettazione;
- casi d'uso$_G$ e scenari principali con flussi e attori associati;
- vincoli tecnici, normativi e di integrazione con sistemi esterni;
- la prioritizzazione dei requisiti$_G$ e l'identificazione del Minimum Viable Product (MVP)$_G$;
- la strategia di tracciabilità$_G$ e gestione delle modifiche (issue tracker, versionamento$_G$ e mappatura requisiti$_G$test$_G$);
- i criteri e i metodi per la verifica$_G$ e la validazione$_G$ dei requisiti$_G$.

Campo	Dettaglio
Redattore$_G$	Analista$_G$
Destinatari	Eggon, Prof. Vardanega, Prof. Cardin
Uso	Esterno

2.1.5.5 Norme di Progetto$_G$

Il documento delle Norme di Progetto$_G$ definisce le metodologie, gli standard e i processi che il team BugBusters adotterà durante lo sviluppo del progetto$_G$ Nexum.

Campo	Dettaglio
Redattore$_G$	Amministratore$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin
Uso	Interno

2.1.5.6 Lettera di Candidatura$_G$

La Lettera di Candidatura$_G$ rappresenta il documento formale con cui il gruppo BugBusters ha ufficializzato la propria candidatura$_G$ per il capitolato$_G$ proposto dall'azienda Eggon. Al suo interno sono sintetizzate le motivazioni che hanno determinato la scelta di questo specifico progetto$_G$, evidenziando i punti di interesse tecnologico e formativo, oltre a fornire i riferimenti al documento di dichiarazione degli impegni e valutazione dei capitolati.

Campo	Dettaglio
Redattore$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin
Uso	Esterno

2.1.5.7 Lettera di Presentazione

Il documento viene redatto in due distinte versioni, corrispondenti alle principali scadenze del progetto$_G$:

- Lettera di Presentazione per la **Requirements and Technology Baseline (RTB)$_G$**
- Lettera di Presentazione per la **Product Baseline (PB)$_G$**

Campo	Dettaglio

Redattore$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin
Uso	Esterno

2.1.5.8 Verbali

I verbali sono documenti di lavoro indispensabili per tracciare le decisioni, le discussioni e le azioni concordate durante le riunioni interne del team e le riunioni con la proponente$_G$ Eggon. Si dividono in verbali interni, prodotti durante le riunioni del team, e verbali esterni, redatti dopo gli incontri con la proponente$_G$.

Campo	Dettaglio interni	Dettaglio esterni
Redattore$_G$	Responsabile$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno	Esterno

2.1.5.9 Piano di Progetto$_G$

Il Piano di Progetto$_G$ rappresenta il documento di riferimento per la pianificazione strategica e operativa delle attività del gruppo BugBusters. Esso definisce la scansione temporale del lavoro suddiviso in sprint$_G$, l'allocazione delle risorse umane e strumentali, nonché l'analisi preventiva dei rischi$_G$ con le relative strategie di mitigazione$_G$. Il documento ha inoltre la funzione fondamentale di monitorare l'andamento del progetto$_G$, riportando a ogni avanzamento il consuntivo$_G$ delle ore produttive e dei costi sostenuti rispetto a quanto preventivato, garantendo così il pieno controllo sulle risorse impiegate.

Campo	Dettaglio interni	Dettaglio esterni
Redattore$_G$	Responsabile$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno	Esterno

2.1.5.10 Piano di Qualifica$_G$

Il Piano di Qualifica$_G$ documenta la strategia adottata dal gruppo BugBusters per garantire la qualità$_G$ del prodotto$_G$ software che il gruppo deve realizzare e dei processi correlati. Esso definisce nel dettaglio le metodologie di verifica$_G$ e validazione$_G$, gli standard di qualità$_G$ di riferimento e le metriche utilizzate per il monitoraggio. Il documento riporta inoltre la pianificazione e gli esiti dei test$_G$ effettuati, con l'obiettivo specifico di assicurare il rispetto dei requisiti$_G$ e mantenere una copertura del codice (code coverage$_G$) pari o superiore all'80%.

Campo	Dettaglio interni	Dettaglio esterni
Redattore$_G$	Responsabile$_G$	Responsabile$_G$
Destinatari	BugBusters, Prof. Vardanega, Prof. Cardin	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno	Esterno

2.1.5.11 Specifica Tecnica

La Specifica Tecnica$_G$ documenta l'architettura di sistema e le scelte progettuali adottate dal gruppo BugBusters per lo sviluppo del prodotto$_G$ software. Esso definisce nel dettaglio i design pattern scelti, lo stack tecnologico impiegato, l'architettura ad alto e basso livello, e il tracciamento tra i componenti software e i requisiti. Il documento funge da linea guida operativa per i programmatori, con l'obiettivo specifico di garantire una solida base strutturale che favorisca la scalabilità$_G$, la manutenibilità$_G$ del codice e la corretta implementazione$_G$ delle funzionalità$_G$.

Campo	Dettaglio
Redattore$_G$	Progettisti
Destinatari	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno ed Esterno

2.1.5.12 Manuale Utente

Il Manuale Utente$_G$ documenta le istruzioni operative e le linee guida necessarie per il corretto utilizzo del prodotto$_G$ software rilasciato dal gruppo BugBusters. Esso definisce nel dettaglio i requisiti minimi di sistema, le procedure di configurazione, e illustra passo-passo tutte le funzionalità$_G$ dell'applicativo divise per tipologia di utente (ruoli). Il documento riporta inoltre il supporto visivo delle interfacce grafiche e la gestione degli errori più comuni, con l'obiettivo specifico di rendere l'utente finale completamente autonomo nell'utilizzo del sistema e minimizzarne la curva di apprendimento.

Campo	Dettaglio
Redattore$_G$	Amministratore$_G$
Destinatari	BugBusters, Eggon, Prof. Vardanega, Prof. Cardin
Uso	Interno ed Esterno

2.2 Processi di sviluppo

Il processo di sviluppo definisce le attività tecniche e operative necessarie per la realizzazione del prodotto$_G$ software che il gruppo deve realizzare, in conformità ai requisiti$_G$ stabiliti durante la fase di fornitura. Questo processo include la progettazione, l'implementazione$_G$, il testing e la documentazione del software, garantendo che ogni fase sia eseguita secondo standard di qualità$_G$ elevati e best practice$_G$ del settore.

2.2.1 Attività previste

- **Implementazione$_G$ del processo**: se non definito prima, il fornitore dovrebbe definire o scegliere un modello di ciclo di vita del software appropriato alla complessità del progetto$_G$;
- **Analisi dei Requisiti$_G$ di sistema**: l'uso previsto del sistema da sviluppare deve essere analizzato per definire i requisiti$_G$ di sistema. La specifica dei requisiti$_G$ di sistema deve descrivere le funzioni e le capacità del sistema, i requisiti$_G$ di business, organizzativi e degli utenti, nonché i requisiti$_G$ relativi a sicurezza$_G$, protezione, fattori umani (ergonomia), interfacce, operatività e manutenzione. Devono inoltre essere inclusi i vincoli di progettazione e i requisiti$_G$ di qualificazione;
- **Design architetturale del sistema**: deve essere definita un'architettura di alto livello del sistema, che identifichi gli elementi hardware, software e le operazioni manuali. Tutti i requisiti$_G$ di sistema devono essere correttamente assegnati a tali elementi. A partire da questi, devono essere individuati gli elementi di configurazione hardware, software e le operazioni manuali;
- **Analisi dei Requisiti$_G$ del software**: il fornitore deve stabilire e documentare i requisiti$_G$ del software, includendo caratteristiche di qualità$_G$: come le caratteristiche funzionali, i requisiti$_G$ di sicurezza$_G$ e di interfaccia$_G$ esterna del software;
- **Design architetturale del software**: il fornitore deve tradurre i requisiti$_G$ dell'elemento software in un'architettura che ne descriva la struttura di alto livello e identifichi i componenti software. Tutti i requisiti$_G$ devono essere assegnati ai componenti software e ulteriormente dettagliati per supportare la progettazione di dettaglio;
- **Design di dettaglio del software**: il fornitore deve sviluppare il progetto$_G$ di dettaglio per ciascun componente software. I componenti devono essere ulteriormente suddivisi in unità software implementabili, compilabili e testabili. Tutti i requisiti$_G$ software devono essere correttamente assegnati dalle componenti alle unità software;
- **Scrittura e test$_G$ del codice**: il fornitore deve scrivere il codice per tutte le componenti individuate, e testarle esaustivamente;

- **Integrazione del software**: il fornitore deve sviluppare un piano di integrazione per assemblare le unità e i componenti software nel prodotto$_G$ software finale. Il piano deve includere i requisiti$_G$ di test$_G$, le procedure, i dati necessari, le responsabilità e la pianificazione delle attività;
- **Test$_G$ di qualifica del software**: il fornitore deve eseguire i test$_G$ di qualificazione in conformità ai requisiti$_G$ di qualificazione del software. Deve essere verificato che l'implementazione$_G$ di ogni requisito$_G$ software sia testata per garantirne la conformità;
- **Integrazione del sistema**: gli elementi di configurazione software devono essere integrati con quelli hardware, con le operazioni manuali e con altri sistemi, se necessario, all'interno del sistema complessivo. Gli insiemi risultanti devono essere testati progressivamente rispetto ai requisiti$_G$ previsti;
- **Testi di qualifica del sistema**: devono essere eseguiti i test$_G$ di qualificazione del sistema in conformità ai requisiti$_G$ di qualificazione definiti. Deve essere verificato che l'implementazione$_G$ di ogni requisito$_G$ di sistema sia testata per garantirne la conformità e che il sistema sia pronto per la consegna;
- **Installazione del software**: il fornitore deve predisporre un piano per l'installazione del prodotto$_G$ software nell'ambiente di destinazione previsto dal contratto. Devono essere definiti e resi disponibili le risorse e le informazioni necessarie all'installazione. Secondo quanto stabilito dal contratto, il fornitore deve supportare il committente$_G$ nelle attività di configurazione iniziale. Qualora il nuovo software sostituisca un sistema esistente, il fornitore deve supportare l'esercizio in parallelo, se richiesto;
- **Supporto per l'approvazione$_G$ del software**: il fornitore deve supportare il committente$_G$ nella revisione e nei test$_G$ di accettazione del prodotto$_G$ software. Le attività di accettazione devono tenere conto dei risultati delle revisioni congiunte, degli audit, dei test$_G$ di qualificazione del software e, se eseguiti, dei test$_G$ di qualificazione del sistema;

2.2.2 Analisi dei Requisiti$_G$

È un attività fondamentale della Requirements and Technology Baseline (RTB)$_G$ e consiste nella raccolta, analisi e documentazione di tutti i requisiti$_G$ che il sistema sviluppato dovrà soddisfare. Tale attività è documentata nell'apposito documento di Analisi dei Requisiti$_G$, che servirà poi come riferimento per guidare tutte le fasi successive di progettazione, implementazione$_G$ e testing del software. Nella sezione seguente viene illustrato il sistema di nomenclatura adottato per identificare in modo univoco i casi d'uso$_G$.

2.2.2.1 Casi d'uso$_G$

UC-SezionePrincipale.Secondario

dove:

- **UC**: sta per Use Case$_G$ (Caso d'Uso$_G$);
- **SezionePrincipale**: abbiamo identificato quattro moduli principali identificati da un numero da 0 a 3;
- **Principale**: è identificato da una lettera dell'alfabeto, e rappresenta un caso d'uso primario non correlato ad altri casi d'uso$_G$, se non tramite inclusione;

- **Secondario**: è identificato da un numero, ed è correlato esclusivamente a un caso d'uso principale;

Ad esempio, il caso d'uso "UC-1A.1" rappresenta un caso d'uso secondario (1) correlato al caso d'uso principale "UC-1A", che a sua volta è associato alla sezione principale 1 del sistema. Ogni caso d'uso viene anche descritto da un nome che lo riassume e da una descrizione dettagliata.

2.2.2.2 Requisiti$_G$

I requisiti$_G$ vengono individuati dopo aver identificato i casi d'uso$_G$, e sono classificati nel modo seguente:

Tipologia-Numero

dove:

- **Tipologia**: rappresenta la categoria del requisito$_G$, e può essere funzionale (RF), requisiti$_G$ di qualità$_G$ (RQ), requisiti$_G$ di vincolo$_G$ (RV) o requisiti$_G$ prestazionali$_G$ (RP);
- **Numero**: è un numero progressivo che identifica in modo univoco il requisito$_G$ all'interno della sua tipologia.

Ad ogni requisito$_G$ viene poi associata una descrizione dettagliata, la fonte del requisito$_G$ (interna, di capitolato$_G$, da colloquio con la proponente$_G$, o da Use Case$_G$), e la priorità$_G$ (Obbligatorio$_G$, Desiderabile$_G$, Opzionale$_G$).

2.2.3 Progettazione

2.2.3.1 Descrizione e scopo

La fase di progettazione rappresenta un passaggio fondamentale nello sviluppo software, in quanto consente di trasformare i requisiti emersi durante l'analisi in una soluzione strutturata e ben definita dal punto di vista architetturale.
In questa fase vengono stabilite le caratteristiche principali del sistema e i vincoli da rispettare, facilitando così l'organizzazione del lavoro e la successiva implementazione$_G$ del codice.
Prima di procedere con la progettazione dettagliata, viene generalmente svolta un'attività preliminare basata sulla realizzazione di un prototipo iniziale, spesso sotto forma di Proof of Concept. Questo artefatto, pensato per essere temporaneo, ha lo scopo di verificare la fattibilità tecnica delle soluzioni ipotizzate e di ridurre eventuali incertezze legate alle tecnologie da utilizzare.
Attraverso questa sperimentazione iniziale è possibile individuare gli strumenti più adatti e, in collaborazione con la proponente$_G$, delineare le funzionalità$_G$ essenziali che costituiranno la prima versione del prodotto$_G$, ovvero il Minimum Viable Product$_G$.

2.2.3.2 Qualità$_G$ dell'architettura

Per una buona architettura software si richiedono le seguenti qualità$_G$:

- **Sufficienza**: L'architettura assicura la piena soddisfazione dei requisiti funzionali$_G$ e non funzionali$_G$ stabiliti per il sistema.
- **Comprensibilità**: La struttura architetturale è chiara e facilmente interpretabile da tutti gli stakeholder$_G$, agevolando manutenzione ed evoluzione.

- **Modularità$_G$**: Il sistema è suddiviso in moduli autonomi e ben definiti, ciascuno sviluppabile, testabile e aggiornabile indipendentemente.
- **Robustezza**: L'architettura gestisce correttamente input imprevisti e condizioni di errore, mantenendo comportamenti coerenti.
- **Flessibilità**: La progettazione consente adattamenti a nuovi requisiti o tecnologie senza interventi strutturali estesi.
- **Riusabilità**: Componenti e moduli sono riutilizzabili in contesti differenti, riducendo duplicazioni e aumentando l'efficienza$_G$.
- **Efficienza$_G$**: L'architettura ottimizza l'uso delle risorse per garantire prestazioni elevate con consumo contenuto.
- **Affidabilità$_G$**: Il software opera in modo coerente e prevedibile nel tempo, rispettando le funzionalità$_G$ attese.
- **Disponibilità**: Il sistema è progettato per essere operativo e accessibile quando richiesto, minimizzando i periodi di inattività.
- **Sicurezza$_G$ rispetto a malfunzionamenti**: L'architettura contiene meccanismi che permettono al sistema di continuare a funzionare anche in caso di guasti.
- **Sicurezza$_G$ rispetto a intrusioni**: Sono previste protezioni contro accessi non autorizzati, garantendo riservatezza e integrità dei dati.
- **Semplicità**: La soluzione è mantenuta il più semplice possibile, riducendo complessità e facilitando comprensione e manutenzione.
- **Incapsulazione**: I dettagli implementativi sono nascosti dietro interfacce ben definite, esponendo solo ciò che è necessario.
- **Coesione**: Ogni modulo$_G$ presenta componenti che collaborano per uno scopo comune, evitando responsabilità frammentate.
- **Basso accoppiamento**: Le dipendenze tra moduli sono minimizzate per migliorare manutenibilità$_G$ e capacità di evolvere il sistema.

2.2.3.3 UML Classi

I diagrammi delle classi UML vengono utilizzati per rappresentare la struttura statica del sistema, descrivendo le entità principali, i loro attributi e le relazioni che intercorrono tra di esse. Questa rappresentazione consente di avere una visione chiara dell'organizzazione interna del software e facilita la comprensione delle responsabilità assegnate a ciascuna classe.
Inoltre, i diagrammi supportano la fase di sviluppo, fungendo da riferimento per l'implementazione$_G$ e la manutenzione del codice.
Verrà adottato lo standard UML 2.

2.2.3.4 C4

Il modello C4 viene adottato per descrivere l'architettura del sistema a diversi livelli di dettaglio, offrendo una visione progressiva che parte dal contesto generale fino ad arrivare ai componenti interni. Questo approccio permette di comunicare in modo efficace la struttura del sistema sia a stakeholder$_G$ tecnici che non tecnici. In particolare, vengono utilizzati diagrammi di contesto, contenitori e componenti per evidenziare le interazioni tra le varie parti del sistema e le tecnologie coinvolte.

2.2.3.5 Design Pattern

Durante la progettazione vengono impiegati opportuni design pattern al fine di adottare soluzioni consolidate a problemi ricorrenti. L'utilizzo di questi schemi progettuali contribuisce a migliorare la qualità$_G$ del codice, rendendolo più modulare, riutilizzabile e facilmente estendibile. La scelta dei pattern avviene in base alle esigenze specifiche del progetto$_G$, con l'obiettivo di garantire una struttura solida e manutenibile nel tempo.
Si pone una particolare attenzione ad usare design pattern ove ci siano problemi e siano effettivamente utili, evitando di introdurre complessità non necessaria. Inoltre si cerca di adottare pattern che non siano nocivi per le tecnologie utilizzate, e che siano ben supportati dagli strumenti di sviluppo scelti.
Viste le scelte architetturali adottate, ci sono molte scelte vincolate (opinionated), e quindi non è possibile adottare pattern che non siano compatibili con tali scelte.

2.2.4 Codifica

2.2.4.1 Descrizione e scopo

La fase di codifica consiste nella traduzione concreta delle soluzioni progettuali in codice sorgente. In questa attività, i componenti definiti durante la progettazione vengono implementati utilizzando i linguaggi e le tecnologie scelte. L'obiettivo principale è realizzare un prodotto$_G$ funzionante che rispetti i requisiti stabiliti, mantenendo al contempo chiarezza, coerenza e qualità$_G$ del codice. Una buona fase di codifica contribuisce a ridurre errori, semplificare le fasi successive di test$_G$ e manutenzione, e garantire una maggiore affidabilità$_G$ del sistema.

2.2.4.2 Best practice di codifica

Per assicurare un elevato standard qualitativo, durante la codifica vengono adottate una serie di buone pratiche.
Tra queste rientrano l'utilizzo di convenzioni di naming coerenti, la scrittura di codice leggibile e ben strutturato, e la suddivisione in moduli con responsabilità ben definite.
È inoltre importante limitare la duplicazione del codice, favorire il riuso e mantenere una documentazione aggiornata dove necessario.
Ulteriori accorgimenti includono l'uso di sistemi di versionamento$_G$ per tracciare le modifiche, la revisione del codice tra pari (code review) e l'esecuzione di test$_G$ durante lo sviluppo.
Queste pratiche permettono di individuare tempestivamente eventuali problemi e di migliorare progressivamente la qualità$_G$ complessiva del prodotto$_G$.

3 Processi di Supporto

L'obiettivo dei processi di supporto è fornire le risorse, gli strumenti e le metodologie necessarie per supportare efficacemente i processi primari di sviluppo e fornitura del software. I processi di supporto sono essenziali per garantire che il team segua pratiche di lavoro coerenti e sia sempre allineato agli obiettivi che si prefigge di raggiungere. Per esempio la documentazione svolge un ruolo cruciale nel mantenere la tracciabilità$_G$ delle decisioni, eliminare qualsiasi ambiguità nella comprensione dei requisiti$_G$ tra i membri del team, e assicurare che tutte le informazioni rilevanti siano facilmente accessibili e aggiornate.

3.1 Documentazione

3.1.1 Scopo

Lo scopo del processo di documentazione è definire le linee guida e le metodologie per la creazione, gestione e manutenzione della documentazione di progetto$_G$. Questo processo mira a garantire che tutta la documentazione prodotta sia chiara, coerente, accessibile e aggiornata, facilitando la comunicazione tra i membri del team, la proponente$_G$ e gli stakeholder$_G$ esterni. Una documentazione ben strutturata supporta l'efficace gestione del progetto$_G$, la tracciabilità$_G$ delle decisioni e la conformità agli standard di qualità$_G$ previsti.

3.1.2 Strumenti Utilizzati

Per supportare le attività di documentazione, il team BugBusters utilizza i seguenti strumenti:

- **GitHub$_G$**: Per la gestione collaborativa della documentazione, il versionamento$_G$ dei documenti e la produzione asincrona. Vengono utilizzate le funzionalità$_G$ di issue$_G$ tracking e project board per tracciare le attività di documentazione. Inoltre BugBusters fa ampio uso dei branch per garantire che le modifiche alla documentazione siano revisionate prima di essere integrate nella versione principale.
- **LaTeX$_G$**: Per la stesura di documenti tecnici e formali, garantendo un formato professionale e coerente. È stata stabilita un'identità visiva comune per tutti i documenti, inclusi template$_G$, stili e convenzioni di formattazione.
- **Viewer PDF sviluppato dal team**: Per facilitare la consultazione della documentazione prodotta.

3.1.3 Documenti Prodotti

Di seguito l'elenco dei documenti obbligatori, coerenti con le indicazioni del capitolato$_G$ C4. Ogni documento verrà mantenuto aggiornato e versionato sul repository$_G$ del progetto$_G$.

1. **Norme di Progetto$_G$**: Il presente documento, dove vengono definite linee guida metodologiche, standard di lavoro e convenzioni adottate dal team.
2. **Piano di Progetto$_G$**: Documento dettagliato con allocazione delle risorse umane e materiali per le milestone$_G$ e le consegne.
3. **Piano di Qualifica$_G$**: Strategia e metriche di collaudo; obiettivo minimo di qualità$_G$: copertura dei test$_G$ pari o superiore al 80%.
4. **Analisi dei Requisiti$_G$**: Specifica dei requisiti$_G$ funzionali$_G$ e non funzionali$_G$, casi d'uso$_G$ principali e criteri di accettazione.
5. **Glossario$_G$**: Elenco e definizioni dei termini tecnici e degli acronimi usati nel progetto$_G$.
6. **Lettera di Presentazione**: Documento di presentazione formale rivolto a RTB$_G$, completo e firmato, da utilizzare per la consegna iniziale.
7. **Verbali (interni/esterni)**: Registrazione degli incontri e delle decisioni, sia per le riunioni interne sia per i confronti con la proponente$_G$.

Tutti i documenti saranno sottoposti a revisione interna e tracciati tramite GitHub$_G$ (issue$_G$ e pull request) per garantirne la tracciabilità$_G$ e la storicizzazione.

3.1.4 Struttura di un documento

Un documento generalmente, esclusi verbali e diario di bordo, ha questa struttura:

- **Copertina**: Logo e informazioni del team, titolo del documento, redattori, verificatori, versione, data di ultima modifica, destinatari.
- **Registro delle modifiche**: Tabella che traccia le versioni del documento, le modifiche apportate, le date e i responsabili.
- **Indice**: Elenco delle sezioni e sottosezioni con i numeri di pagina.
- **Introduzione**: Scopo del documento, scopo del prodotto$_G$, glossario$_G$ e riferimenti.
- **Corpo del documento**: Contenuto principale, suddiviso in sezioni e sottosezioni.
- **Appendici**: Materiale supplementare, come diagrammi, tabelle o esempi.

3.1.4.1 Struttura dei verbali

I verbali seguono una struttura semplificata:

- **Copertina**: Logo e informazioni del team, titolo del documento, redattore$_G$, verificatore$_G$, versione, data, uso e destinatari.
- **Abstract**: Elenco degli argomenti da trattare durante la riunione.
- **Indice**: Elenco delle sezioni e sottosezioni con i numeri di pagina.
- **Informazioni Generali**: Data, ora, luogo, partecipanti e assenti.
- **Ordine del Giorno**: Pianificazione degli argomenti da discutere.
- **Svolgimento**: Dettagli delle discussioni ed eventuali argomenti fuori programma.
- **Tabella delle decisioni e delle azioni**: elenco strutturato delle decisioni prese, con descrizione e incaricato/responsabile.
- **Esito Riunione**: Sintesi dei risultati e delle conclusioni.

Da questo schema è possibile notare che i verbali non hanno il registro delle modifiche. Questo perché sono considerati documenti di lavoro e non documenti formali che richiedono una tracciabilità$_G$ delle versioni.

Dal momento che il ruolo del responsabile$_G$ costituisce il maggior dispendio di budget all'interno del progetto$_G$ e che la redazione dei verbali è una delle sue principali responsabilità, è stato deciso di adottare una struttura standardizzata per i verbali al fine di ottimizzare il tempo impiegato nella loro redazione, introducendo anche delle macro per LaTeX$_G$ che permettono di velocizzare ulteriormente la creazione dei verbali.

3.1.4.2 Struttura dei Diari di Bordo

I diari di bordo sono diapositive presentate durante la lezione settimanale di Ingegneria del Software dedicata alla discussione e condivisione delle difficoltà e dubbi incontrati durante lo sviluppo del progetto$_G$. Per questo motivo i diari di bordo hanno pochi punti:

- **Copertina**: Logo e informazioni del team, titolo del documento.

- **Difficoltà incontrate**: Lista delle difficoltà incontrate durante la settimana.
- **Dubbi**: Elenco di domande e dubbi da porre ai docenti.

Visto che i diari di bordo sono da presentare in pochi minuti non hanno bisogno di una struttura complessa, bensì di essere sintetici e diretti per permettere a docenti e colleghi di capire velocemente quali sono i problemi riscontrati.

3.1.5 Denominazione dei documenti

I documenti prodotti dal team BugBusters seguono una convenzione di denominazione standardizzata per garantire coerenza, facilità di identificazione e tracciabilità$_G$.

La struttura del nome dei **verbali** è la seguente:

`<TIPO>_<DATA>.pdf`

Dove <TIPO> può essere:

- **VI**: Verbale Interno$_G$
- **VE**: Verbale Esterno$_G$

E <DATA> è la data della riunione nel formato `GG-MM-AAAA`.
Esempio: `VE_15-01-2026.pdf` rappresenta il verbale esterno$_G$ della riunione del 15 gennaio 2026.

La struttura del nome dei **diari di bordo** è la seguente:

`DB-<DATA>.pdf`

Dove <DATA> è la data di presentazione del diario nel formato `GG-MM-AAAA`.
Esempio: `DB-22-01-2026.pdf` rappresenta il diario di bordo presentato il 22 gennaio 2026.

Per gli altri **documenti formali** vengono utilizzati nomi descrittivi chiari e concisi del documento stesso, ad esempio Norme di Progetto$_G$.pdf, Piano di Progetto$_G$.pdf, Analisi dei Requisiti$_G$.pdf ecc.

3.1.6 Produzione

La produzione della documentazione segue un processo strutturato per garantire qualità$_G$, coerenza e tracciabilità$_G$.

- **Pianificazione**: A seconda del documento, viene pianificata la sua creazione in base alle milestone$_G$ del progetto$_G$ e alle esigenze del team. Secondo il tipo di documento, viene assegnato un redattore$_G$ e un verificatore$_G$ secondo i ruoli definiti nel way of working$_G$ (vedi sezione 2.1.5).
- **Creazione del branch di lavoro**: Per ogni documento viene creato un branch dedicato nel repository$_G$ GitHub$_G$ per permettere la collaborazione e il versionamento$_G$. Per verbali e diari di bordo non vengono creati branch per ogni singolo documenti bensì sono stati creati dei branch "verbali" e "diari-di-bordo" che raccolgono rispettivamente tutti i verbali e tutti i diari di bordo.

- **Redazione**: Il redattore$_G$ incaricato crea la bozza del documento seguendo le linee guida stabilite nelle Norme di Progetto$_G$, utilizzando gli strumenti appropriati (ad esempio LaTeX$_G$ per documenti formali).
- **Revisione**: Una volta completata la bozza, il documento viene sottoposto al verificatore$_G$ designato che controlla la correttezza, la coerenza e la conformità agli standard di qualità$_G$. Il processo è asincrono e avviene tramite pull request su GitHub$_G$. Se ci sono modifiche vengono tracciate e discusse all'interno della pull request. La verifica$_G$ verrà tracciata all'interno del registro delle modifiche del documento.
- **Approvazione$_G$ e integrazione**: Dopo la revisione, il documento viene approvato dal responsabile$_G$ e integrato nel branch principale del repository$_G$ attraverso una pull request. Viene aggiornato il registro delle modifiche per riflettere la versione finale.
- **Modifiche**: Qualsiasi modifica successiva al documento segue lo stesso processo di redazione, revisione e approvazione$_G$, garantendo che tutte le versioni siano tracciate e documentate.

3.2 Gestione della configurazione

La gestione delle configurazioni è l'insieme di attività e strumenti che servono a controllare, tracciare e organizzare tutte le modifiche fatte a un qualsiasi elemento del progetto$_G$.
Basandosi sullo standard ISO/IEC 12207:1995, la gestione della configurazione deve occuparsi di monitorare le modifiche e le versioni degli elementi del sistema, tenere traccia del loro stato e delle richieste di cambiamento, assicurare che gli elementi siano completi, coerenti e corretti, e controllare come vengono archiviati, spostati e consegnati.

3.2.1 Strumenti utilizzati

Per ognuna delle attività previste, BugBusters utilizza:

- **Git**: come sistema di controllo di versione distribuito per tracciare le modifiche al codice sorgente e ai documenti.
- **GitHub$_G$**: servizio di hosting per repository$_G$ Git utilizzato per il versionamento$_G$ di codice e documentazione. Fornisce strumenti per la collaborazione (branching, pull request e code review), per il tracciamento delle attività (issues$_G$, project board) e per l'integrazione continua (Actions). Su GitHub$_G$ vengono centralizzate le risorse di progetto$_G$ e registrate tutte le modifiche, garantendo controllo degli accessi e tracciabilità$_G$.

3.2.2 Attività previste

L'attività di gestione delle configurazioni svolta dal gruppo si avvale delle seguenti attività:

- **Identificazione della configurazione**: l'identificazione delle componenti che formeranno il prodotto$_G$ da sviluppare
- **Controllo della configurazione**: il controllo dei cambiamenti con opportuni metodi di approvazione$_G$ e rifiuto degli stessi;
- **Registrazione dello stato di configurazione**: il come rappresentare la storia dei cambiamenti subiti da ciascun elemento sviluppato;
- **Valutazione della configurazione**: come determinare l'efficacia$_G$ del prodotto$_G$ sviluppato, ossia la sua conformità ai requisiti;

3.2.3 Identificazione della configurazione

Data la complessità architetturale del progetto$_G$ definito nel capitolato$_G$, l'identificazione della configurazione risulta un'attività fondamentale. In fase di progettazione, le varie parti del sistema verranno pertanto schematizzate e documentate in un artefatto specifico, da redigere e approvare durante la Product Baseline (PB)$_G$.

3.2.4 Controllo della configurazione

Il controllo della configurazione consiste nell'amministrare le richieste di modifica che vengono poi approvate o meno.
Per tracciare le modifiche da approvare BugBusters utilizza le **issue$_G$** la **board** e le **pull request** predisposte da GitHub$_G$ nel modo seguente:

- **Issue$_G$**: ogni modifica da apportare viene documentata mediante l'apertura di una issue$_G$ relativa, quest'ultima viene assegnata a un componente che la prenderà in carico e procederà con le modifiche al relativo documento o codice.

Ogni issue$_G$ è identificata da un codice univoco dato dalla fase del progetto$_G$ a cui essa è relativa, e da un numero incrementale (es: RTB1,RTB2,RTB3 ecc...).
Per velocizzare il processo di gestione delle issue$_G$, BugBusters utilizza una GitHub action che, se in qualsiasi commit viene scritto: *chiudi <NOME ISSUE>*, la issue$_G$ corrispondente verrà immediatamente chiusa, qualsiasi sia il branch in cui ci si trova.

- **Board**: Serve per definire lo stato in cui si trova una issue$_G$, ovvero se: è ancora da iniziare, è in sviluppo o è terminata. Il team ha scelto il modello Kanban per la gestione delle issue$_G$.
- **Pull Request**: serve per richiedere la verifica$_G$ o approvazione$_G$ di una modifica prima di fonderla con un ramo della repository$_G$. Il team BugBusters, oltre che per un corretto workflow$_G$ di verifica$_G$, usa le pull request come unico modo di modifica in produzione, in quanto si vuole che il branch main sia modificato solo a seguito di una pull request con 2 review positive.

3.2.5 Registrazione stato di configurazione

Per poter tenere traccia dei cambiamenti di ogni documento e codice, è previsto il sistema di versionamento$_G$ seguente:

X.Y.Z

Dove:

- **X**: subisce un incremento solo quando il file viene approvato;
- **Y**: subisce un incremento solo quando il file viene verificato;
- **Z**: subisce un incremento quando viene fatto un qualsiasi tipo di modifica al file;

3.2.6 Valutazione della configurazione

La valutazione della configurazione verifica$_G$ il perfetto allineamento del software ai requisiti e al design architetturale, garantendo tramite il Tracciamento dei Requisiti il rilascio di un prodotto$_G$ efficiente (pienamente conforme alle specifiche) e sufficiente (privo di complessità superflue). Per

assicurare tale aderenza durante l'intero ciclo di sviluppo, il gruppo adotterà una strategia integrata: il codice verrà esplicitamente commentato per tracciare il legame diretto con il requisito$_G$ soddisfatto, mentre il testing automatizzato e le metriche di code coverage$_G$ permetteranno di validare costantemente le funzionalità$_G$ e individuare tempestivamente eventuali aree scoperte. A supporto di ciò, la rigorosa modularità$_G$ del sistema separerà le responsabilità architetturali, favorendo una tracciabilità$_G$ netta e lineare tra ogni singolo componente e i requisiti di riferimento.

3.3 Accertamento della Qualità$_G$

Il processo di Accertamento della Qualità$_G$ (o *Quality Assurance*) ha lo scopo di garantire che le metodologie adottate e i prodotti realizzati dal gruppo BugBusters soddisfino i requisiti minimi di efficienza$_G$ ed efficacia$_G$ prestabiliti. Per assicurare tale livello qualitativo, il gruppo fa riferimento a due standard internazionali fondamentali: lo standard **ISO/IEC 12207:1995** per la gestione e la qualità$_G$ dei processi e lo standard **ISO/IEC 9126** per la misurazione della qualità$_G$ del prodotto$_G$ software. L'accertamento si concretizza operativamente attraverso continue e rigorose attività di Verifica$_G$ e Validazione$_G$.

3.3.1 Strumenti e metriche a supporto

Per valutare oggettivamente l'avanzamento e la bontà del lavoro svolto, BugBusters si avvale di un insieme definito di metriche quantificabili, suddivise tra metriche di processo e metriche di prodotto$_G$. L'impiego di indicatori puramente oggettivi garantisce un'analisi fredda, imparziale e basata sui dati.

Tutte le metriche adottate, complete di descrizione, valore di accettazione e valore ottimo, sono formalizzate nel *Piano di Qualifica*$_G$ (Sezione "Obiettivi stabiliti per la qualità$_G$"). Il monitoraggio di tali indicatori avviene tramite l'aggiornamento del **Cruscotto$_G$ di Valutazione**, uno strumento che permette al team di analizzare visivamente i trend storici, individuare precocemente eventuali deviazioni (es. calo della *Code Coverage*$_G$ o indici di costo sotto soglia) e applicare tempestivamente le necessarie azioni correttive (ciclo PDCA).

3.3.2 Attività previste

Le attività previste per garantire il corretto svolgimento del processo di Accertamento della Qualità$_G$ si articolano nei seguenti punti:

- **Implementazione$_G$ del processo:** definizione rigorosa degli standard normativi di riferimento e delle procedure operative necessarie per la misurazione, la raccolta e l'analisi dei dati;
- **Accertamento della qualità$_G$ di processo:** monitoraggio continuo dei processi primari, organizzativi e di supporto (es. stabilità dei requisiti, pianificazione, gestione dei costi) per verificare la stretta aderenza al *way of working*$_G$ stabilito e misurare l'efficienza$_G$ temporale ed economica del team;
- **Accertamento della qualità$_G$ di prodotto$_G$:** validazione$_G$ del software rilasciato rispetto alle caratteristiche di Funzionalità$_G$, Affidabilità$_G$, Efficienza$_G$, Usabilità, Manutenibilità e Portabilità. Tale attività è supportata da una campagna di testing automatizzato e manuale (Integrazione, Sistema, Accettazione) per garantire la conformità ai requisiti.

Al fine di mantenere un controllo costante sullo stato di salute del progetto$_G$, le misurazioni descritte vengono effettuate e consolidate al termine di ogni Sprint$_G$. I risultati raccolti vengono

quindi documentati e storicizzati nel *Piano di Qualifica*$_G$, certificando la qualità$_G$ del lavoro e supportando le decisioni strategiche intraprese dal team.

3.4 Qualifica

3.4.1 Verifica$_G$

Il processo di Verifica$_G$ ha come obiettivo principale quello di verificare che quanto prodotto$_G$ sia a regola d'arte, ovvero conforme con i requisiti$_G$ richiesti.
L'obiettivo della verifica$_G$ è poter rispondere alla domanda «Did I build the system right?» gli esiti di tale processo sono riportati nel documento di Piano di Qualifica$_G$ nella sezione dedicata ai test$_G$.
Notare che al momento della condivisione di questo documento al pubblico, il team BugBusters avrà solamente raggiunto l'RTB$_G$, per questo motivo lo stato dei test$_G$ sara "Non Implementato". Al raggiungimento della PB$_G$, questa sezione verrà aggiornata con i risultati dei test$_G$ eseguiti.

3.4.1.1 Attività di verifica$_G$

La verifica$_G$ agisce su singoli segmenti di sviluppo, accertando che l'esecuzione di essi non abbia introdotto errori. In questa prospettiva, un'attività di verifica$_G$ è il controllo che il prodotto$_G$ ottenuto al termine di una fase sia congruente con il semilavorato avuto come punto di partenza di quella fase.
Il team BugBusters si è principalmente occupato della verifica$_G$ relativa alla documentazione, controllando la correttezza grammaticale, sintattica e la correttezza del contenuto di ogni documento.
Relativamente alle verifiche sul codice, sarà un argomento che verrà trattato più in dettaglio dopo il raggiungimento della *Requirements and Technology Baseline*$_G$.
In ogni caso tutte le informazioni relative alla verifica$_G$, verranno riportate nel Piano di Qualifica$_G$.

3.4.1.2 Analisi Statica

L'analisi statica non richiede l'esecuzione dell'oggetto di cui si fa la verifica$_G$, ciò permette di applicarla già prima della fine della codifica.
Può essere eseguita mediante **metodi formali** (ad esempio prove matematiche) o mediante **metodi di lettura**. Tra i **metodi di lettura** ne troviamo due significativi:

- **Walkthrough**: si esegue un esame privo di assunzioni o presupposti, non si sa esattamente dove è più probabile che vi siano difetti, dunque si guarda ovunque. Si percorre il codice simulandone possibili esecuzioni e si studia ogni parte di documento come farebbe un compilatore, verificando che le regole siano soddisfatte. È un metodo di verifica$_G$ costoso e non automatizzabile;
- **Inspection**: si esegue un esame focalizzato su presupposti, si sa già dove cercare, i verificatori dunque rilevano la presenza di difetti eseguendo una lettura mirata dell'oggetto di verifica$_G$. Non è esaustiva come il Walkthrough, ma permette di creare una lista di controllo apposita per ogni oggetto di verifica$_G$, così da individuare potenziali problemi;

Ogni passo documenta attività svolte e risultanze. Gli errori identificati vengono discussi tra verificatore$_G$ e autore mentre le modifiche vengono apportate generalmente solo dall'autore.

3.4.1.3 Analisi Dinamica

L'analisi dinamica è chiamata così perché per essere eseguita necessita l'esecuzione dell'oggetto da verificare.
Serve per verificare se sono presenti comportamenti inattesi, e dunque rimuovere o modificare le parti che ne causano il fault.
Per raggiungere tale obiettivo, si usufruisce di Test$_G$, che devono essere ripetibili e automatizzabili.

- **Ripetibili**: poiché se si presenta un failure nel codice, e vengono corretti i fault, posso eseguire lo stesso test$_G$ nelle stesse condizioni per verificare l'effettiva correzione del failure. Perché un test$_G$ sia ripetibile, l'ambiente di esecuzione (e quindi lo stato iniziale) deve essere sempre lo stesso;
- **Automatizzabili**: poiché i test$_G$ possono essere centinaia, è necessario automatizzarli mediante driver (che serve per pilotare il test$_G$), stub (che simula i moduli necessari al test$_G$ ma che non ne sono oggetto) e logger (che registra ciò che avviene durante l'esecuzione).

Le principali tipologie di Test$_G$ sono:

- **Test$_G$ di Unità$_G$**;
- **Test$_G$ di Regressione**;
- **Test$_G$ di Integrazione$_G$**;
- **Test$_G$ di Sistema$_G$**;

I test$_G$ eseguiti da BugBusters reperibili nella sezione 3 Metodi di testing del Piano di Qualifica$_G$ sono descritti nella seguente maniera:

- **Codice**: un codice univoco che garantisce la tracciabilità del test$_G$ (quindi la sua identificazione univoca);
- **Descrizione**: una breve descrizione dell'obiettivo del test$_G$;
- **Requisito$_G$**: codice univoco del requisito$_G$ a cui il test$_G$ fa riferimento. Tale codice è reperibile nell'Analisi dei Requisiti$_G$ nella sezione 3.1 Requisiti$_G$ funzionali$_G$;
- **Stato**: lo stato del test$_G$, che può essere:
  - **Non Implementato**: il test$_G$ non è ancora stato implementato;
  - **Superato**: il test$_G$ è stato implementato ed eseguito;
  - **Non Superato**: il test$_G$ è stato implementato ed eseguito, ma non ha raggiunto i criteri di accettazione;
  - **Da Eseguire**: il test$_G$ è stato implementato, ma non è ancora stato eseguito.

3.4.1.4 Test$_G$ di Unità

I **test$_G$ di unità$_G$** hanno lo scopo di verificare le singole unità di un software, definite durante la progettazione di dettaglio. Per unità si intende la più piccola quantità di Software che sia utilmente sottoponibile a verifica$_G$ come oggetto singolo.
I test$_G$ di unità possono essere funzionali (black-box), cioè basati sulle specifiche dell'unità. In questo caso, vengono verificati gli input e output dati dal sistema, ma non viene verificata la

logica che fornisce tali risultati.
Tuttavia, i test$_G$ funzionali da soli non sono sufficienti per verificare la correttezza della logica interna del modulo$_G$. Per questo motivo vengono affiancati dai test$_G$ strutturali (white-box), che analizzano la logica interna dell'unità cercando di percorrere tutti i cammini di esecuzione al suo interno, raggiungendo una copertura massima.
L'esecuzione di questi test$_G$ può essere facilitata dall'uso del debugger.
Il testing di unità si considera completo quando tutte le unità del sistema sono state verificate.

3.4.1.5 Test$_G$ di Regressione

I **Test$_G$ di regressione** sono necessari affinché delle modifiche effettuate per aggiunta, correzione o rimozione non pregiudichino le funzionalità$_G$ già verificate.
Per far ciò il test$_G$ di regressione comprende tutti i test$_G$ necessari ad accertare che la modifica di una parte P $\in$ S, non causi errori in P o in alcuna altra parte di S esterna a P.

3.4.1.6 Test$_G$ di Integrazione

I **Test$_G$ di integrazione$_G$** verificano il corretto funzionamento di più unità software combinate tra loro, controllando che le loro interazioni avvengano come previsto.
L'obiettivo è assicurarsi che le unità già testate singolarmente comunichino e collaborino correttamente tra loro.
Ci sono diverse strategie per implementare tali test$_G$:

- **Bottom-up**: si parte dalle componenti più basse (con meno dipendenze) e si integrano progressivamente verso l'alto, usando driver per simulare moduli superiori mancanti;
- **Top-down**: si dalle componenti di alto livello (con più dipendenze) e si scende verso quelli inferiori, usando stub per simulare moduli non ancora sviluppati;

3.4.1.7 Test$_G$ di Sistema

I **Test$_G$ di sistema$_G$** verificano il comportamento dell'intero software nel suo complesso, controllando che il sistema soddisfi i requisiti$_G$ funzionali$_G$ e non funzionali$_G$ specificati.

3.4.2 Validazione$_G$

La **validazione$_G$** è un processo per confermare in modo definitivo che le caratteristiche del software siano conformi ai bisogni dell'utente finale e all'uso previsto, risponde alla domanda: «Did I build the right system?»

3.4.2.1 Processo di Validazione$_G$

BugBusters ha raccolto tutte le richieste di **Eggon** e le ha documentate nell'Analisi dei Requisiti$_G$, in modo tale da poter tracciare correttamente i requisiti$_G$ e da poter eseguire i test$_G$ di accettazione per controllare che quanto sviluppato corrisponda alle richieste di **Eggon**.

I risultati della validazione$_G$ verranno riportati nel documento di Piano di Qualifica$_G$ nella sezione Test$_G$ di accettazione.

Si fa notare che al momento della condivisione di questo documento al pubblico, il team BugBusters avrà solamente raggiunto l'RTB$_G$, per questo motivo lo stato dei test$_G$ sara "Non implementato". Al raggiungimento della PB$_G$, questa sezione verrà aggiornata con i risultati dei test$_G$.

4 Processi Organizzativi

4.1 Descrizione e Scopo

Secondo lo standard ISO/IEC 12207:1997, il processo di gestione include tutte le attività necessarie a portare a termine un progetto$_G$ software garantendo il conseguimento degli obiettivi prefissati nel rispetto dei requisiti$_G$, dei tempi e dei costi.
Per raggiungere questi risultati il processo si basa su:

- una pianificazione accurata delle attività, delle milestone$_G$ e delle risorse;
- il monitoraggio continuo dell'avanzamento mediante indicatori e report periodici;
- la gestione dei rischi$_G$ e delle modifiche attraverso procedure di controllo e di escalation;
- l'adozione tempestiva di azioni correttive ogni volta che gli scostamenti rispetto al piano lo richiedono;
- la definizione chiara di ruoli, responsabilità e canali di comunicazione fra gli stakeholder$_G$.

In sintesi, il processo di gestione assicura che il progetto$_G$ sia eseguito in modo controllato e tracciabile, consentendo decisioni informate e interventi rapidi per mantenere il progetto$_G$ allineato ai vincoli di qualità$_G$, tempi e costi.

4.2 Attività

4.2.1 Gestione degli Sprint$_G$

Ogni due settimane viene pianificato uno sprint$_G$ in cui vengono assegnate le attività da svolgere ai vari membri del team. Viene fatto un meeting di pianificazione in cui si scelgono le attività da svolgere e si assegnano ai vari membri. Al termine dello sprint$_G$ viene fatta una review in cui si mostrano i risultati ottenuti e si discutono le difficoltà incontrate. Viene anche fatto un meeting di retrospettiva$_G$ in cui si discute su cosa è andato bene e cosa può essere migliorato per il prossimo sprint$_G$. Durante lo sprint$_G$ vengono definiti i vari ruoli che i membri del team devono svolgere.

4.3 Ruoli

4.3.1 Responsabile$_G$

Il responsabile$_G$ ha il compito di coordinare il team, pianificare le attività, gestire le risorse e comunicare con la proponente$_G$ e i docenti. In particolare dovrà assegnare i compiti, redigere i documenti:

- Verbali Esterni ed Interni, con assegnazione dei compiti previsti nelle riunioni.
- Diario di Bordo
- Avanzamento Piano di Progetto$_G$

- Redigere il documento di Piano di Qualifica$_G$;

4.3.2 Amministratore$_G$

È la figura incaricata di sviluppare, mantenere e migliorare gli strumenti, le risorse e i processi che garantiscono il regolare avanzamento del progetto$_G$. Si occupa di supervisionare la configurazione degli ambienti di lavoro e di monitorare le scadenze di carattere amministrativo, offrendo supporto al team nelle varie attività. Tra le sue responsabilità rientrano:

- Predisporre e gestire gli ambienti di lavoro;
- Controllare e aggiornare gli strumenti utilizzati dal gruppo per collaborare e comunicare;
- Occuparsi del versionamento$_G$ dei documenti;
- Redigere e mantenere aggiornato il documento Norme di Progetto$_G$.

4.3.3 Analista$_G$

L'analista$_G$ ha il compito di raccogliere, analizzare e documentare i requisiti$_G$ del progetto$_G$, assicurandosi che siano chiari, completi e coerenti con le esigenze del proponente$_G$. Inoltre, collabora con il team per tradurre i requisiti$_G$ in specifiche tecniche utilizzabili durante le fasi di progettazione e sviluppo. In particolare dovrà occuparsi di:

- Redigere il documento di Analisi dei Requisiti$_G$;
- Gestire le richieste di chiarimento e modifica dei requisiti$_G$ con la proponente$_G$;
- collaborare con il team per garantire che i requisiti$_G$ siano compresi e implementati correttamente.

4.3.4 Progettista$_G$

Il progettista$_G$ ha il compito di definire l'architettura e la struttura del sistema, assicurandosi che soddisfi i requisiti$_G$ funzionali$_G$ e non funzionali$_G$ stabiliti. Collabora con il team per tradurre i requisiti$_G$ in soluzioni tecniche efficienti e scalabili. In particolare dovrà occuparsi di:

- Determinare le tecnologie e gli strumenti più adatti per lo sviluppo del progetto$_G$;
- Definire l'architettura del sistema e le sue componenti principali;
- Supervisionare lo sviluppo tecnico e garantire la coerenza con le specifiche progettuali;

4.3.5 Programmatore$_G$

Il programmatore$_G$ ha il compito di implementare le funzionalità$_G$ del sistema secondo le specifiche tecniche fornite dal progettista$_G$. Collabora con il team per garantire che il codice sia di alta qualità$_G$, efficiente e mantenibile. In particolare dovrà occuparsi di:

- scrivere il codice sorgente seguendo le linee guida e gli standard di programmazione stabiliti, traducendo le specifiche tecniche definite dal progettista$_G$ in soluzioni funzionanti;
- Occuparsi del testing del codice per garantire la correttezza e l'affidabilità$_G$ delle funzionalità$_G$ implementate;
- Collaborare con il team per risolvere problemi tecnici e implementare nuove funzionalità$_G$.

4.3.6 Verificatore$_G$

Il verificatore$_G$ ha il compito di assicurare che il prodotto$_G$ sviluppato soddisfi i requisiti$_G$ stabiliti e che sia di alta qualità$_G$. Collabora con il team per identificare e risolvere problemi, garantendo che il sistema sia affidabile, efficiente e conforme agli standard. In particolare dovrà occuparsi di:

- pianificare e condurre attività di verifica$_G$, come test$_G$ funzionali, test$_G$ di integrazione e test$_G$ di sistema;
- revisionare la documentazione prodotta per garantire la correttezza e la completezza;
- documentare i risultati delle attività di verifica$_G$ e segnalare eventuali difetti o non conformità riscontrate;
- identificare possibili miglioramenti o punti critici nel processo di sviluppo e proporre soluzioni per aumentarne l'efficacia$_G$.

4.4 Tracciamento Ore

Per garantire una corretta gestione del tempo e delle risorse, ogni membro del team BugBusters è tenuto a registrare le ore lavorate su ciascuna attività. Il tracciamento avviene tramite un foglio di calcolo condiviso su Google Sheets, dove ogni membro del team inserisce le ore dedicate alle varie attività e alle relative issue$_G$ di progetto$_G$. Questo approccio centralizzato consente di monitorare l'avanzamento effettivo rispetto alla pianificazione, identificare tempestivamente discrepanze tra stima e consuntivo$_G$ e facilitare decisioni di riallocazione delle risorse, se necessario.

Il foglio di calcolo contiene inoltre la tabella di stima iniziale, permettendo una comparazione immediata tra preventivo$_G$ e consuntivo$_G$ e garantendo visibilità sull'andamento complessivo del progetto$_G$. Ogni membro del team è responsabile$_G$ dell'accuratezza e della propria registrazione.

4.5 Tracciamento Azioni

Le azioni sono tracciate nei verbali delle riunioni interne ed esterne. Dopo ogni riunione il responsabile$_G$ redige il verbale$_G$ in cui vengono riportate tutte le decisioni prese e le azioni da svolgere. Ogni azione viene assegnata a un membro del team tramite issue$_G$ tracking su GitHub$_G$, dunque nei verbali è presente il collegamento con il ticket aperto sulla piattaforma. Nel caso ci fossero problemi o dubbi riguardo l'azione assegnata, il membro del team può aprire una discussione all'interno della issue$_G$ per chiedere chiarimenti o segnalare difficoltà. Come riportato dalla sezione 3.2, il team utilizza automazioni per rendere più fluido il processo di tracciamento delle azioni, garantendo maggior tracciamento possibile.

4.6 Gestione dei Rischi

L'individuazione e la gestione dei rischi$_G$ di progetto$_G$ si articola in tre diverse fasi descritte di seguito:

- **Identificazione**: al fine di prevenire problemi futuri si vanno ad individuare le parti critiche del progetto$_G$ analizzando ogni potenziale rischio$_G$ associato. Per ogni rischio$_G$ individuato occorre specificare il nome, una descrizione del rischio$_G$, il livello di probabilità$_G$ e l'impatto. Le tipologie di rischio$_G$ e le relative tabelle di dettaglio sono descritte nel *Piano di Progetto$_G$*. Ad ogni rischio$_G$ è inoltre associato un codice univoco con il seguente formato:

R-[Tipologia]-[ID]

Le sigle utilizzate per la classificazione della tipologia sono:

- **G**: Rischio$_G$ Globale$_G$;
- **T**: Rischio$_G$ Tecnologico$_G$;
- **I**: Rischio$_G$ Individuale$_G$.

L'ID è un valore numerico incrementale a una cifra.

- **Mitigazione$_G$**: per ogni rischio$_G$ individuato si definiscono a priori le possibili contromisure da attuare nel caso in cui il problema si presenti, al fine di ridurne l'impatto sul progetto$_G$.
- **Monitoraggio**: i rischi individuati vengono costantemente monitorati dal Responsabile$_G$ durante l'intero ciclo di vita del progetto$_G$, al fine di individuare tempestivamente la loro occorrenza e applicare le relative contromisure stabilite.

4.7 Comunicazione

4.7.1 Comunicazione Interna

Le comunicazioni interne al team BugBusters avvengono principalmente tramite:

- **WhatsApp**: utilizzato per comunicazioni rapide, discussioni tecniche e coordinamento quotidiano tra i membri del team.
- **Discussioni GitHub$_G$**: utilizzate per comunicazioni formali relative a issue$_G$, pull request e documentazione, garantendo tracciabilità$_G$ e storicizzazione delle decisioni. Utili in quanto permettono di comunicare solo con i membri interessati alla discussione.

In caso di problemi tecnici o necessità di confronto più approfondito, il team può organizzare riunioni virtuali tramite piattaforme Discord$_G$. Per ulteriori dettagli sulle riunioni, si veda la sezione 4.8.

4.7.2 Comunicazione Esterna

Le comunicazioni esterne con la proponente$_G$ e i docenti avvengono principalmente tramite:

- **Email**: utilizzata per comunicazioni formali, invio di documenti e aggiornamenti sullo stato del progetto$_G$, usando la mail bugbusters.unipd@gmail.com.
- **Telegram**: utilizzato per comunicazioni rapide e coordinamento con la proponente$_G$.

Analogamente per quanto riguarda le comunicazioni interne, in caso di necessità di confronto più approfondito, il team può organizzare riunioni virtuali tramite piattaforma Google Meet$_G$. Per ulteriori dettagli sulle riunioni, si veda la sezione 4.8.

4.8 Riunioni

4.8.1 Riunioni Interne

Le riunioni interne al team BugBusters sono pianificate regolarmente per garantire un coordinamento efficace e un avanzamento costante del progetto$_G$.

È previsto un incontro settimanale, solitamente il lunedì, in cui si discute lo stato di avanzamento del progetto$_G$, si pianificano le attività della settimana e si affrontano eventuali problematiche riscontrate.
Inoltre, vengono organizzate riunioni ad hoc in caso di necessità, ad esempio per discutere questioni specifiche o per risolvere problemi urgenti.
Le riunioni si svolgono principalmente tramite piattaforma Discord$_G$, ma possono essere organizzate anche in presenza se necessario.
Al termine di ogni riunione, viene redatto un verbale$_G$ che riassume le decisioni prese e le azioni da intraprendere, garantendo tracciabilità$_G$ e responsabilità all'interno del team.

4.8.2 Riunioni Esterne

Le riunioni esterne con la proponente$_G$ sono pianificate per garantire un allineamento continuo sugli obiettivi del progetto$_G$ e per ricevere feedback preziosi.
Queste riunioni si svolgono generalmente ogni due settimane, il mercoledì alle ore 15:00, ma possono essere organizzate con maggiore frequenza in caso di necessità.
Questi incontri avvengono tramite piattaforma Google Meet$_G$. Al termine di ogni riunione, viene redatto un verbale$_G$ che riassume le decisioni prese e le azioni da intraprendere, il quale verrà condiviso con la proponente$_G$ e, se in linea con quanto discusso, firmato.

4.9 Miglioramento

4.9.1 Descrizione e Scopo

Il processo di miglioramento ha come obiettivo l'analisi, la misurazione, il controllo e l'ottimizzazione dell'intero ciclo di vita del software. La finalità principale è assicurare che il prodotto$_G$ risponda pienamente ai requisiti e alle aspettative, garantendo allo stesso tempo alti livelli di qualità$_G$ ed efficienza$_G$.
Attraverso un approccio iterativo, ogni fase viene riesaminata con regolarità e progressivamente affinata, consentendo di individuare e applicare interventi specifici orientati al continuo perfezionamento del prodotto$_G$.

4.9.2 Ciclo di miglioramento continuo

Il processo di miglioramento si articola in quattro fasi principali. La prima consiste nell'individuazione dei processi organizzativi da adottare nelle varie attività di progetto$_G$ lungo tutto il ciclo di vita del software. Tali processi devono essere opportunamente documentati e affiancati da un sistema di controllo che ne permetta la gestione, il monitoraggio e l'evoluzione nel tempo.
La seconda fase riguarda l'applicazione concreta dei miglioramenti individuati, garantendo che le modifiche vengano integrate in modo coerente nei processi già esistenti. È fondamentale aggiornare la documentazione per rispecchiare le variazioni introdotte, mentre le informazioni storiche, tecniche e di valutazione devono essere raccolte e analizzate al fine di sostenere un progresso costante.
La terza fase è dedicata alla verifica$_G$ dei processi, effettuata sulla base degli obiettivi stabiliti, delle metriche definite e dei dati raccolti. Inoltre, la pianificazione e lo svolgimento di revisioni periodiche assicurano che i processi rimangano adeguati ed efficaci nel tempo.
Infine, la quarta fase mira a consolidare e standardizzare i miglioramenti ottenuti, favorendo un'evoluzione continua dei processi e riducendo il rischio$_G$ di regressioni. Questo approccio iterativo consente di mantenere nel tempo elevati livelli di qualità$_G$ ed efficienza$_G$.

4.10 Formazione

4.10.1 Descrizione e Scopo

La formazione ha lo scopo di garantire che ogni componente del gruppo acquisisca le competenze necessarie per affrontare in modo efficace e consapevole le possibili criticità che possono emergere durante lo sviluppo del progetto$_G$.
Il percorso di apprendimento viene strutturato a partire da un'analisi approfondita delle esigenze individuate, a cui segue la selezione delle risorse e degli strumenti più idonei per il raggiungimento degli obiettivi prefissati. In questo modo, il gruppo può sviluppare progressivamente le proprie conoscenze, rafforzando al contempo la padronanza delle tecnologie e delle metodologie adottate.
La formazione rappresenta quindi un elemento essenziale per la buona riuscita del progetto$_G$, contribuendo non solo alla qualità$_G$ del risultato finale, ma anche alla crescita professionale e personale di tutti i membri coinvolti.

4.10.2 Formazione individuale

Nel corso dello sviluppo del progetto$_G$, ciascun componente del gruppo è chiamato a sviluppare in modo autonomo le competenze necessarie all'utilizzo delle tecnologie coinvolte. Questo apprendimento non avviene in modo isolato, ma è supportato dal confronto continuo tra i membri del team, in cui chi possiede maggiore esperienza contribuisce a guidare e chiarire eventuali difficoltà.
Un ruolo importante è svolto anche dalla consultazione di fonti esterne, come la documentazione ufficiale delle tecnologie utilizzate, risorse online e materiali di approfondimento. Inoltre, la realizzazione di piccoli progetti o esercitazioni pratiche consente di consolidare le conoscenze acquisite e di sperimentare direttamente le soluzioni studiate.
Questo approccio collaborativo e proattivo permette di uniformare il livello di preparazione del gruppo, rendendolo adeguato al raggiungimento degli obiettivi prefissati e favorendo al tempo stesso un avanzamento più efficace e consapevole dell'intero progetto$_G$.

5 Qualità$_G$ del Software

Per qualità$_G$ si intende l'insieme delle caratteristiche di un'entità che ne determinano la capacità di soddisfare esigenze sia espresse che implicite. La qualità$_G$ si misura su due aspetti principali:

1. la **qualità$_G$ del prodotto$_G$**;
2. la **qualità$_G$ del processo**.

La qualità$_G$ del prodotto$_G$ finale è una conseguenza diretta della qualità$_G$ del processo utilizzato per crearlo. Perché il software risulti di qualità$_G$ è necessario che venga pianificato, e non solo controllato, un buon processo di valutazione: questo permette di lavorare seguendo una modalità proattiva (valutazione continua) piuttosto che reattiva (correzione di bug).
Diventa quindi fondamentale seguire uno standard di riferimento per poter definire, misurare e valutare la qualità$_G$ del software in modo oggettivo e coerente.

5.1 Standard di riferimento e modelli

Storicamente definite dagli standard ISO/IEC 9126 e ISO/IEC 14598, queste sono oggi unificate e aggiornate nella serie di norme ISO/IEC 25010, nota anche come SQuaRE (Systems and

software Quality Requirements and Evaluation). Questo standard internazionale organizza la qualità$_G$ in 2 diverse categorie:

- Qualità$_G$ del Prodotto$_G$, che analizza le seguenti proprietà intrinseche del software:
  1. **Adeguatezza funzionale**;
  2. **Efficienza$_G$ prestazionale**;
  3. **Compatibilità**;
  4. **Usabilità**;
  5. **Affidabilità$_G$**;
  6. **Sicurezza$_G$**;
  7. **Manutenibilità$_G$**;
  8. **Portabilità**;
- Qualità$_G$ in Uso, che valuta le seguenti caratteristiche dal punto di vista dell'utente finale:
  1. **Efficacia$_G$**;
  2. **Efficienza$_G$**;
  3. **Soddisfazione**;
  4. **Libertà da rischi$_G$**;
  5. **Contesto di copertura**.

Noi ci concentriamo sulla qualità$_G$ del prodotto$_G$.
Per quanto riguarda, invece, la qualità$_G$ del processo, lo standard di riferimento è l'**ISO/IEC 9001** e il **modello CMMI (Capability Maturity Model Integration)**. Questi standard forniscono linee guida per la gestione della qualità$_G$ nei processi di sviluppo software, includendo aspetti come la pianificazione, il controllo e il miglioramento continuo dei processi stessi.

5.2 Processo di Valutazione della Qualità$_G$

Il processo di valutazione della qualità$_G$ del software si articola in quattro fasi principali:

1. **Fase preventiva di definizione di metriche, interpretazione delle misure e criteri di accettazione**: sia nel contesto del prodotto$_G$ che del processo, vengono identificate le metriche e come queste devono essere interpretate per valutare la qualità$_G$ e raggiungere l'obiettivo di accettazione.
2. **Misurazione**: la raccolta dei dati necessari per calcolare le metriche selezionate. Questa fase dovrebbe essere automatizzata attraverso strumenti di analisi statica e dinamica del codice, test$_G$ automatici e monitoraggio delle prestazioni.
3. **Valutazione**: l'analisi dei dati raccolti per determinare se gli obiettivi di qualità$_G$ sono stati raggiunti. Questa fase può includere la revisione dei risultati da parte di un team di verifica$_G$ e validazione$_G$, nonché la documentazione dei risultati e delle eventuali azioni correttive da intraprendere.
4. **Accettazione**: la decisione finale su se il software soddisfa i requisiti$_G$ di qualità$_G$ stabiliti. Se le metriche sono soddisfatte, il software può essere considerato pronto.

Questo intero processo è documentato all'interno del Piano di Qualifica$_G$.

5.3 Principi della Qualità$_G$ del processo

Secondo lo standard ISO/IEC 9001, la qualità$_G$ del processo di sviluppo software si basa su diversi principi fondamentali:

- **Responsabilità della direzione**: il responsabile$_G$ deve definire una politica per la qualità$_G$, stabilire obiettivi e garantire che tutti i membri del team siano consapevoli delle loro responsabilità.
- **Gestione delle risorse**: è essenziale assicurarsi che il team abbia gli strumenti giusti (es. IDE, versionamento$_G$ con git) e che i membri abbiano la formazione necessaria.
- **Realizzazione del prodotto$_G$**: il processo di sviluppo richiede che le fasi di documentazione dei requisiti$_G$, progettazione e sviluppo siano sottocontrollo e tracciabili.
- **Misurazione, analisi e miglioramento**: è fondamentale monitorare continuamente il processo di sviluppo per identificare aree di miglioramento e implementare azioni correttive quando necessario.

Ciò viene fatto in questo progetto$_G$ attraverso una chiara definizione degli strumenti e delle metodologie di lavoro (specificate in questo documento), un monitoraggio continuo dell'avanzamento del progetto$_G$ (tramite issue$_G$ tracking e riunioni periodiche documentate), un'attenta analisi sulle richieste della proponente$_G$ (specificate nel documento di Analisi dei Requisiti$_G$) e una valutazione regolare della qualità$_G$ del prodotto$_G$ con un'analisi sul miglioramento (tramite il processo di verifica$_G$ e validazione$_G$ descritto nel Piano di Qualifica$_G$).

5.4 Metriche per la qualità$_G$

Come descritto nella sezione precedente, la qualità$_G$ del software viene misurata attraverso l'uso di metriche specifiche. Queste metriche forniscono dati quantitativi che aiutano a valutare diversi aspetti della qualità$_G$ del prodotto$_G$ e del processo di sviluppo.
Le metriche selezionate per questo progetto$_G$ vengono definite qui e misurate durante il processo di valutazione della qualità$_G$ descritto nel Piano di Qualifica$_G$.
Le metriche sono suddivise in due categorie principali:

1. **Metriche per la qualità$_G$ del prodotto$_G$**: queste metriche valutano aspetti come la funzionalità$_G$, l'affidabilità$_G$, l'efficienza$_G$, l'usabilità, la manutenibilità$_G$ e la portabilità del software.
2. **Metriche per la qualità$_G$ del processo**: queste metriche monitorano l'efficacia$_G$ del processo di sviluppo, inclusi tempi di consegna, tassi di difetti, copertura dei test$_G$ e conformità agli standard di qualità$_G$.

5.4.1 Qualità$_G$ del Processo

5.4.1.1 Processi primari

5.4.1.2 Fornitura

MPC01 **Earned Value (EV)$_G$**: misura il valore del lavoro effettivamente completato rispetto al costo preventivato.

$$EV_G = BAC \times \text{Percentuale di completamento}$$

dove BAC (Budget at Completion) è il budget totale previsto per il progetto$_G$.

MPC02 Planned Value (PV)$_G$: misura il valore del lavoro pianificato fino a una certa data.

$$PV_G = BAC \times \text{Percentuale di pianificazione}$$

Interpretazione: Risponde alla domanda: «Quanto lavoro avremmo dovuto completare fino a questa data?».
Valore ideale: per ogni sprint$_G$ il valore deve essere <= EAC$_G$

MPC03 Actual Cost (AC)$_G$: misura il costo effettivo sostenuto per il lavoro completato fino a una certa data (che coincide con quella di fine sprint$_G$).

MPC04 Cost Performance Index (CPI)$_G$: indica l'efficienza$_G$ dei costi del progetto$_G$.

$$CPI_G = \frac{EV_G}{AC_G}$$

Interpretazione: Risponde alla domanda: «Quanto lavoro è stato completato rispetto a quanto è stato speso?».

Se $CPI_G > 1$, il progetto$_G$ è sotto budget (positivo).

MPC05 Schedule Performance Index (SPI)$_G$: indica l'efficienza$_G$ del tempo del progetto$_G$.

$$SPI_G = \frac{EV_G}{PV_G}$$

Interpretazione: Risponde alla domanda: «Quanto lavoro è stato effettivamente completato rispetto a quanto pianificato?».

Se $SPI_G > 1$, il progetto$_G$ è in anticipo rispetto alla pianificazione (positivo).

MPC06 Estimated at Completion (EAC$_G$): stima il costo totale del progetto$_G$ alla sua conclusione, basandosi sulle prestazioni attuali.

$$EAC_G = \frac{BAC}{CPI_G}$$

Interpretazione: Risponde alla domanda: «Quanto costerà il progetto$_G$ se le attuali prestazioni di costo continuano?».
Valore ideale = BAC

MPC07 Estimate to Complete (ETC)$_G$: stima il costo necessario per completare il lavoro rimanente del progetto$_G$.

$$ETC_G = EAC_G - AC_G$$

Interpretazione: Risponde alla domanda: «Quanto costerà completare il progetto$_G$?».
Valore ideale <= EAC$_G$

MPC08 Time Estimate At Completion (TEAC): stima il tempo totale necessario per completare il progetto$_G$.

$$TEAC = \frac{\text{Durata Pianificata}}{SPI_G}$$

dove SPI_G (Schedule Performance Index$_G$) è $\frac{EV_G}{PV_G}$. *Interpretazione*: Risponde alla domanda: «Quanto tempo ci vorrà per completare il progetto$_G$ se le attuali prestazioni di tempo continuano?».
Valore ideale <= Durata pianificata.

5.4.1.3 Sviluppo

MPC09 Requirements Stability Index (RSI)$_G$: misura la stabilità dei requisiti$_G$ durante il ciclo di vita del progetto$_G$.

$$RSI_G = \left(1 - \frac{N_{agg} + N_{mod} + N_{canc}}{N_{iniz}}\right) \times 100$$

dove:

- N_{agg} è il numero di nuovi requisiti$_G$ emersi dopo l'inizio del periodo;
- N_{mod} è il numero di requisiti$_G$ modificati;
- N_{canc} è il numero di requisiti$_G$ rimossi;
- N_{iniz} è il numero di requisiti$_G$ definiti all'inizio del periodo.

Interpretazione: un valore più alto indica una maggiore stabilità dei requisiti$_G$.

5.4.1.4 Processi di supporto

5.4.1.5 Documentazione

MPC10 Indice di Gulpease$_G$: misura la leggibilità di un testo in lingua italiana, utile per valutare la documentazione tecnica.

$$\text{Indice di Gulpease}_G = 89 + \frac{300 \times \text{Numero di frasi} - 10 \times \text{Numero di lettere}}{\text{Numero di parole}}$$

Interpretazione: Un valore più alto indica una maggiore leggibilità del testo

5.4.1.6 Verifica$_G$

MPC11 Code Coverage$_G$: misura la percentuale di codice sorgente coperto dai test$_G$.

$$\text{Code Coverage}_G = \frac{\text{Linee di codice testate}}{\text{Linee di codice totali}} \times 100\%$$

Interpretazione: Un valore più alto indica una maggiore copertura dei test$_G$, riducendo il rischio$_G$ di bug non rilevati.

MPC12 Test$_G$ Success Rate: misura la percentuale di test$_G$ superati con successo.

$$\text{Test}_G \text{ Success Rate} = \frac{\text{Numero di test}_G \text{ superati}}{\text{Numero totale di test}_G \text{ eseguiti}} \times 100\%$$

Interpretazione: Un valore più alto indica una maggiore affidabilità$_G$ del software.

5.4.1.7 Gestione della qualità$_G$

MPC13 Quality metrics satisfied: misura la percentuale di metriche di qualità$_G$ soddisfatte rispetto a quelle definite nel Piano di Qualifica$_G$.

$$\text{Quality metrics satisfied} = \frac{\text{Numero di metriche soddisfatte}}{\text{Numero totale di metriche}} \times 100\%$$

Interpretazione: Un valore più alto indica una maggiore conformità agli standard di qualità$_G$ stabiliti.

5.4.1.8 Processi organizzativi

5.4.1.9 Gestione dei processi

MPC14 Time efficiency: misura l'efficienza$_G$ nella gestione del tempo rispetto alla pianificazione.

$$\text{Time efficiency} = \frac{\text{Ore produttive}}{\text{Ore totali}} \times 100\%$$

Interpretazione: Un valore più alto indica una migliore gestione del tempo e rispetto delle scadenze.

5.4.2 Qualità$_G$ del Prodotto$_G$

5.4.2.1 Adeguatezza funzionale

MPD01 Requisiti$_G$ obbligatori soddisfatti.
Valore ideale: 100

MPD02 Requisiti$_G$ desiderabili soddisfatti.
Valore ideale: 100 *(accettato 0)*

MPD03 Requisiti$_G$ opzionali soddisfatti.
Valore ideale: 100 *(accettato 0)*

MPD04 AI$_G$ Acceptance Rate (rating $\geq 3/5$).
Valore ideale: $\geq 80\%$ *(accettato* $\geq 60\%$*)*

5.4.2.2 Affidabilità$_G$

MPD05 Branch Coverage$_G$.
Valore di accettazione: $\geq 70\%$
Valore ideale: $\geq 85\%$

MPD06 Defect Density.
Valore di accettazione: ≤ 3 difetti / KLOC
Valore ideale: ≤ 1 difetto / KLOC

5.4.2.3 Efficienza$_G$

MPD07 UI Response Time (Interfaccia$_G$).
Valore di accettazione: ≤ 2 sec
Valore ideale: ≤ 0.5 sec

MPD08 Code Response Time - Ai Generativo testo.
Valore di accettazione: ≤ 5 sec
Valore ideale: ≤ 3 sec

MPD09 Code Response Time - Ai Generativo immagini.
Valore di accettazione: ≤ 10 sec
Valore ideale: ≤ 5 sec

MPD10 Code Response Time - Ai Co-Pilot$_G$.
Valore di accettazione: ≤ 10 sec
Valore ideale: ≤ 5 sec

5.4.2.4 Usabilità

MPD11 Click Count (Funzioni principali). Misura il numero di click necessari per accedere alle funzioni principali.
Valore di accettazione: ≤ 5 click
Valore ideale: ≤ 3 click

MPD12 User Error Rate (Errori di validazione$_G$).
Valore di accettazione: $\leq 10\%$
Valore ideale: $\leq 5\%$

5.4.2.5 Manutenibilità$_G$

MPD13 Blocker Code Smells. Misura la presenza di "code smells" di tipo blocker, ovvero problemi nel codice che possono causare gravi malfunzionamenti o difficoltà di manutenzione.
Valore di accettazione: 0
Valore ideale: 0

MPD14 Cyclomatic Complexity$_G$ (per metodo). Misura la complessità ciclomatica$_G$ di un metodo, ovvero il numero di percorsi indipendenti nel codice.
Valore di accettazione: ≤ 15
Valore ideale: ≤ 10

MPD15 Comment Intensity.
Valore di accettazione: $\geq 10\%$
Valore ideale: $\geq 20\%$

5.4.2.6 Portabilità

MPD16 Supported Browsers (Test$_G$ passati).
Valore di accettazione: 100% (Desktop)
Valore ideale: 100% (All devices)