



Norme di progetto

Progetto di Ingegneria del Software A.A. 2025/2026

Versione: 1.0.0

Autore: Atlas

Ultima modifica: 2026/02/06

Tipo di documento: Interno

Registro delle modifiche

Versione	Data	Autore	Verificatore	Descrizione
v1.0.0	2026/02/06	Francesco Marcolongo	Giacomo Giora	Approvazione
v0.8.2	2026/02/06	Andrea Difino	Giacomo Giora	Aggiunte g del glossario
v0.8.1	2026/02/05	Andrea Difino	Giacomo Giora	Correzioni varie
v0.8.0	2026/01/07	Riccardo Valerio	Francesco Marcolongo	Stesura sez. 7
v0.7.2	2025/12/31	Federico Simonetto	Michele Tesser	Fine stesura sez. 5
v0.7.1	2025/12/30	Federico Simonetto	Michele Tesser	Correzioni sez. 3
v0.7.0	2025/12/29	Federico Simonetto	Michele Tesser	Stesura sez. 6
v0.6.0	2025/12/29	Federico Simonetto	Michele Tesser	Inizio stesura sez. 5
v0.5.3	2025/12/29	Federico Simonetto	Michele Tesser	Fine stesura sez. 3
v0.5.2	2025/12/29	Federico Simonetto	Michele Tesser	Fine stesura sez. 4
v0.5.1	2025/12/24	Federico Simonetto	Michele Tesser	Fine stesura sottosez. 3.2
v0.5.0	2025/12/24	Federico Simonetto	Michele Tesser	Inizio stesura sez. 4
v0.4.0	2025/12/23	Federico Simonetto	Michele Tesser	Inizio stesura sottosez. 3.2 e correzione errori
v0.3.1	2025/12/12	Francesco Marcolongo	Michele Tesser	Fine stesura sez. 2
v0.3.0	2025/12/10	Francesco Marcolongo	Riccardo Valerio	Inizio stesura sez. 3
v0.2.0	2025/12/09	Francesco Marcolongo	Riccardo Valerio	Stesura sez. 2
v0.1.0	2025/12/05	Francesco Marcolongo	Bilal Sabic	Prima stesura introduzione

Indice

1 Introduzione	3
1.1 Scopo del documento	3
1.2 Glossario	3
1.3 Riferimenti utili	3
1.3.1 Riferimenti normativi	3
1.3.2 Riferimenti informativi	3
2 Processi Primari	5
2.1 Fornitura	5
2.1.1 Strumenti utilizzati	5
2.1.2 Attività previste	5
2.1.3 Documentazione fornita	6
2.2 Sviluppo	7
2.2.1 Strumenti utilizzati	7
2.2.2 Attività previste	8
2.2.3 Attività di analisi dei requisiti	9
2.2.3.1 Casi d'uso	9
2.2.3.2 Requisiti	9
2.2.4 Codifica del software	10
2.2.4.1 Stile della codifica	10
2.2.4.2 Strumenti e Tecnologie	11
3 Processi di supporto	13
3.1 Documentazione	13
3.1.1 Tipologie di documenti	13
3.1.1.1 Verbali	13
3.1.1.2 Diario di bordo	14
3.1.1.3 Denominazione dei documenti	14
3.1.2 Strumenti utilizzati	15
3.1.3 Attività previste	15
3.1.4 Produzione	15
3.1.5 Manutenzione	16
3.2 Gestione delle configurazioni	16
3.2.1 Strumenti utilizzati	16
3.2.2 Attività previste	16
3.2.3 Identificazione della configurazione	17
3.2.4 Controllo della configurazione	17
3.2.4.1 Pull Request	17
3.2.4.2 Issue	17
3.2.4.3 Project Board	17
3.2.5 Rendicontazione dello stato della configurazione	18

3.2.6	Valutazione della configurazione	18
3.3	Accertamento della qualità	18
3.3.1	Attività previste	18
3.4	Verifica	19
3.4.1	Strumenti utilizzati	19
3.4.2	Attività previste	19
3.4.3	Implementazione del processo	19
3.4.4	Verifica	20
3.4.4.1	Analisi statica	20
3.4.4.2	Analisi dinamica	20
3.4.4.2.1	Test di unità	21
3.4.4.2.2	Test di integrazione	21
3.4.4.2.3	Test di sistema	21
3.4.4.2.4	Test di regressione	22
3.4.4.2.5	Test di accettazione	22
3.5	Validazione	22
3.5.1	Attività previste	22
4	Processi Organizzativi	23
4.1	Gestione dei Processi	23
4.1.1	Attività previste	23
4.1.2	Ruoli e responsabilità	23
4.1.2.1	Responsabile	24
4.1.2.2	Amministratore	24
4.1.2.3	Analista	24
4.1.2.4	Progettista	24
4.1.2.5	Programmatore	24
4.1.2.6	Verificatore	24
4.1.3	Coordinamento e comunicazioni	25
4.1.3.1	Comunicazioni Interne	25
4.1.3.2	Comunicazioni Esterne	25
4.1.3.3	Verbali	25
4.1.4	Gestione delle attività	25
4.1.4.1	Metodologia di lavoro	26
4.1.4.2	Strumenti di Ticketing	26
4.1.4.3	La Board Kanban	26
4.1.4.4	Workflow di un Task	27
4.2	Gestione delle infrastrutture	27
4.2.1	Attività previste	27
4.2.2	Strumenti di supporto	27
4.2.3	Gestione repository	28
4.3	Miglioramento del processo	29

4.3.1	Attività previste	29
4.3.2	Strumenti per il miglioramento	29
4.4	Formazione del personale	29
4.4.1	Attività previste	30
4.4.2	Metodologia di apprendimento	30
4.4.3	Piano di formazione e tempistiche di esecuzione	30
5	Standard ISO/IEC 9126 per la qualità	31
5.1	Funzionalità	31
5.2	Affidabilità	31
5.3	Efficienza	32
5.4	Usabilità	32
5.5	Manutenibilità	32
5.6	Portabilità	33
6	Metriche per la qualità del processo	34
6.1	Processi primari	34
6.1.1	Fornitura	34
6.1.2	Sviluppo	37
6.2	Processi di supporto	38
6.2.1	Documentazione	38
6.2.2	Verifica	38
6.2.3	Qualità	39
6.3	Processi organizzativi	39
6.3.1	Gestione dei processi	39
7	Metriche per la qualità del prodotto	40
7.1	Funzionalità	40
7.2	Affidabilità	41
7.3	Usabilità	42
7.4	Efficienza	42
7.5	Manutenibilità	42

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento *Norme di Progetto* è definire le procedure e gli strumenti di lavoro che il team utilizzerà per tutta la durata del progetto didattico.

Per realizzare la stesura del documento, il team ha preso come riferimento lo standard ISO/IEC 12207:1995. Questo divide i processi di ciclo di vita software in tre categorie:

- **Primari**: processi senza i quali un progetto non può dirsi tale;
- **Supporto**: processi che supportano i processi primari;
- **Organizzativi**: processi di carattere più generale, trasversali rispetto ai singoli progetti.

1.2 Glossario

All'interno della documentazione prodotta dal team possono comparire termini suscettibili di incomprensioni o ambiguità. Per evitare questo, è disponibile un glossario contenente i termini tecnici e le loro definizioni. Un termine è consultabile nel glossario se è indicato con la notazione *parola_G*. Premendo sulla G a pedice, l'utente verrà indirizzato alla pagina web del glossario.

1.3 Riferimenti utili

1.3.1 Riferimenti normativi

- Riferimento al *capitolato_G* 1 dell'azienda proponente:
Bluewind S.r.l - Automated EN18031 Compliance Verification
<https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C1.pdf>
- **Standard ISO/IEC 9126**
https://it.wikipedia.org/wiki/ISO/IEC_9126
- Riferimento al documento dello standard della descrizione dei processi del ciclo di vita software:
ISO/IEC 12207:1995
https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

1.3.2 Riferimenti informativi

- Riferimento alle slide del corso di Ingegneria del Software:
Regolamento del progetto didattico
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/PD1.pdf>
- Riferimento alle slide del corso di Ingegneria del Software:
Processi di ciclo di vita
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T02.pdf>

- Riferimento alle slide del corso di Ingegneria del Software:
Modelli di sviluppo software
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T03.pdf>
- Riferimento alle slide del corso di Ingegneria del Software:
Gestione di progetto
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T04.pdf>
- Riferimento alle slide del corso di Ingegneria del Software:
Verifica e validazione: introduzione
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T09.pdf>
- Riferimento alle slide del corso di Ingegneria del Software:
Verifica e validazione: analisi statica
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T10.pdf>
- Riferimento alle slide del corso di Ingegneria del Software:
Verifica e validazione: analisi dinamica
<https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T11.pdf>

2 Processi Primari

I processi primari comprendono l'insieme di attività che supportano il team di *sviluppo_G* e l'azienda fornitrice nell'avvio e nell'esecuzione dello *sviluppo_G*, nell'operazione e nella manutenzione di un prodotto software.

Sebbene cinque processi siano classificati come primari, in questo documento approfondiremo i processi di **fornitura** e di **sviluppo**, più interessanti dal punto di vista del progetto didattico.

2.1 Fornitura

Il processo di fornitura descrive le attività e gli incarichi del fornitore che hanno il fine di produrre e consegnare il prodotto software. Il processo ha avvio dopo l'aggiudicazione dell'appalto e prosegue con la definizione delle procedure e delle risorse necessarie per gestire e garantire il progetto, comprendendo lo *sviluppo_G* dei piani di progetto e la loro esecuzione, fino alla consegna del prodotto software al committente.

2.1.1 Strumenti utilizzati

Sono stati utilizzati i seguenti strumenti per poter svolgere le attività previste:

- **GitHub:** utilizzato per il sistema di *ticketing_G*, utile per fornire una panoramica generale delle attività previste all'intero team di progetto;
- **Discord:** utilizzato come canale di comunicazione interno;
- **Google Calendar:** utilizzato per tenere traccia delle riunioni, sia interne che esterne;
- **Google Drive:** utilizzato per creare e archiviare i documenti condivisi del gruppo come il foglio delle ore;
- **Google Meet:** utilizzato per partecipare alle riunioni esterne con l'azienda proponente;
- **Telegram:** utilizzato come metodo di comunicazione asincrono con l'azienda proponente.

2.1.2 Attività previste

Il processo di fornitura prevede le seguenti attività:

1. **Avvio:** il fornitore effettua l'analisi delle richieste da parte del proponente, e ne valuta la fattibilità tecnica ed economica tenendo conto delle politiche organizzative e di altre normative applicabili.
2. **Preparazione della risposta:** il fornitore prepara una risposta da inviare alla proponente, includendo le modifiche consigliate emerse durante l'attività di avvio.
3. **Negoziazione:** il fornitore presenta al proponente la risposta formalizzata derivante dall'attività precedente e si avvia una *negoziazione_G* finalizzata alla stipula di un contratto.

4. **Pianificazione:** il fornitore conduce un'attività di analisi e consolidamento dei requisiti finalizzata a stabilire un modello di ciclo di vita del software appropriato allo scopo e alla complessità del prodotto. Contestualmente, vengono definiti i requisiti di qualità che il prodotto dovrà soddisfare e viene effettuata un'analisi dei rischi associati a ciascuna opzione di sviluppo.
5. **Esecuzione e controllo:** il fornitore deve attivare e controllare i processi di *sviluppo G*, esercizio e manutenzione, garantendo la qualità e il rispetto dello stato di avanzamento definito dal ciclo di vita stabilito nel contratto.
6. **Revisione e valutazione:** il fornitore organizza le attività di *revisione G* e accettazione del prodotto insieme al proponente, fornendo i report di valutazione, test e *revisione G* previsti dal contratto.
7. **Consegna e completamento:** il fornitore consegna il prodotto software al proponente, assicurando l'assistenza e il supporto previsti dal contratto.

2.1.3 Documentazione fornita

In questa sottosezione vengono elencati i documenti che il team ha sviluppato e si impegna a consegnare ai professori Vardanega e Cardin e all'azienda proponente *Bluewind S.r.l..*

Analisi dei requisiti	
Redattore	Analisti
Tipo di documento	Esterno
Scopo e contenuti	Il documento riporta l'elenco dei Casi d'Uso (con relativi diagrammi UML) degli attori coinvolti, i requisiti, funzionali e non, collegati ai Casi d'Uso e al capitolato, e le informazioni relative al loro tracciamento.

Tabella 1: Descrizione del documento "Analisi dei requisiti"

Lettera di presentazione	
Redattore	Responsabile
Tipo di documento	Esterno
Scopo e contenuti	Il documento ha come scopo la presentazione ufficiale ai prof. Vardanega e Cardin della candidatura del team alle baseline di progetto.

Tabella 2: Descrizione del documento "Lettera di presentazione"

Norme di progetto	
Redattore	Amministratore
Tipo di documento	Interno
Scopo e contenuti	Documento attuale, riporta gli strumenti e le procedure che il team utilizzerà per tutto lo sviluppo del progetto didattico.

Tabella 3: Descrizione del documento "Norme di progetto"

Piano di progetto	
Redattore	Responsabile
Tipo di documento	Esterno
Scopo e contenuti	Lo scopo del documento è tracciare le attività che vengono svolte così come quelle che vengono pianificate durante tutta la realizzazione del progetto. Il <i>Piano di progetto</i> riporta una descrizione di ogni sprint con l'aggiornamento delle ore utilizzate e i relativi costi.

Tabella 4: Descrizione del documento "Piano di progetto"

Piano di qualifica	
Redattore	Amministratore
Tipo di documento	Esterno
Scopo e contenuti	Il documento riporta le strategie di verifica e di validazione che il team di progetto utilizzerà per garantire la qualità del prodotto e dei processi del progetto, oltre ai test effettuati e i rispettivi esiti.

Tabella 5: Descrizione del documento "Piano di qualifica"

2.2 Sviluppo

Il processo di *sviluppo* si occupa di definire le attività che il team compie per implementare il prodotto software, rispettando le esigenze e le specifiche definite dal proponente.

2.2.1 Strumenti utilizzati

Sono stati utilizzati i seguenti strumenti per poter svolgere le attività previste:

- **Draw.io:** utilizzato per la rappresentazione dei diagrammi degli *Use Case*;
- **GitHub:** utilizzato come VCS per il prodotto software.

2.2.2 Attività previste

Il processo di *sviluppo*_G prevede le seguenti attività:

1. **Implementazione del processo:** lo sviluppatore deve definire e adottare un modello di ciclo di vita del software adeguato agli obiettivi, alla natura e alla complessità del progetto.
2. **Analisi dei requisiti di sistema:** lo sviluppatore deve analizzare il contesto e l'utilizzo previsto del sistema da realizzare, al fine di identificare, definire e documentare in modo completo i requisiti di sistema. Tali requisiti devono includere funzionalità, prestazioni, vincoli operativi e requisiti dell'utente.
3. **Progettazione architetturale del sistema:** lo sviluppatore deve individuare e definire gli elementi hardware e software che compongono il sistema, descrivendone l'*architettura*_G a un livello tale da garantire la soddisfazione di tutti i requisiti identificati nella fase precedente.
4. **Analisi dei requisiti software:** per ciascun elemento software lo sviluppatore deve stabilire e documentare i requisiti software, ovvero i requisiti lato utente. Questi comprendono anche le specifiche di qualità.
5. **Progettazione architetturale del software:** lo sviluppatore deve trasformare i requisiti software in un'*architettura*_G software che descriva la struttura ad alto livello, i componenti e le loro interazioni.
6. **Progettazione in dettaglio del software:** per ogni elemento software, lo sviluppatore deve produrre una progettazione dettagliata, fino a definire le singole unità software.
7. **Codifica e test del software:** lo sviluppatore deve realizzare e testare ogni unità software, verificando che soddisfi i requisiti assegnati.
8. **Integrazione del software:** lo sviluppatore deve predisporre ed eseguire un piano di integrazione per combinare componenti e unità software in un unico elemento software coerente.
9. **Test di qualifica software:** lo sviluppatore deve condurre test di qualifica per ciascun elemento software, assicurando la conformità ai requisiti di qualifica stabiliti.
10. **Integrazione di sistema:** lo sviluppatore deve integrare gli elementi hardware e software in un sistema completo e operativo.
11. **Test di qualifica del sistema:** lo sviluppatore deve condurre test di qualifica dell'intero sistema, verificando la conformità ai requisiti di sistema.
12. **Installazione del software:** lo sviluppatore deve installare il prodotto software nell'ambiente operativo previsto dal contratto o dalla specifica di progetto.

13. **Supporto all'accettazione software:** lo sviluppatore deve supportare l'utente finale durante le attività di verifica, *revisione*_G e test previste per l'accettazione del prodotto.

Il team ha deciso di approfondire di seguito le sole attività di analisi dei requisiti di sistema e codifica del software in vista della *Requirements and Technology Baseline*. Crediamo infatti che, per questa *baseline*_G di progetto, le due attività sopra menzionate siano quelle che richiedono una spiegazione più approfondita. Le altre principali attività che il team presenterà verranno aggiunte dopo il “semaforo verde” della RTB.

2.2.3 Attività di analisi dei requisiti

L'analisi dei requisiti è un'attività cardine di ogni progetto e, in particolare, della *baseline*_G di progetto *Requirements and Technology Baseline*. Questa consiste nell'individuare tutti i requisiti, funzionali e non, che il prodotto software deve soddisfare. L'analisi effettuata dal team può essere visualizzata nell'[apposito documento](#).

In particolare, vengono riportati i casi d'uso ed i relativi requisiti secondo la seguente convenzione.

2.2.3.1 Casi d'uso

I casi d'uso sono identificati secondo il seguente formato: **UCPrincipale.Alternativo** dove:

- **UC** è la sigla di *Use Case*_G, ovvero Caso d'Uso;
- **Principale** è rappresentato da un numero maggiore di zero e identifica il Caso d'Uso principale. Il numero indicato dal campo **Principale** è univoco tra tutti i Casi d'Uso principali;
- **Alternativo**, se presente, è rappresentato da un numero maggiore di zero e indica che il Caso d'Uso è correlato al Caso d'Uso indicato dal campo **Principale**.

Ogni Caso d'Uso è accompagnato dal relativo diagramma *UML*_G, da una descrizione testuale e da un nome che ne riassume lo scenario.

2.2.3.2 Requisiti

I requisiti sono identificati secondo il seguente formato: **R-Id-Tipologia-Priorità** dove:

- **R** sta per *requisito*_G;
- **Id** è un numero maggiore di zero, identificativo univoco di ogni *requisito*_G;
- **Tipologia** indica la tipologia del requisito. È rappresentata da uno di questi tre valori:
 - **F**: funzionale;
 - **Q**: qualità;

- **V**: vincolo.

- **Priorità** indica la priorità del requisito. È rappresentata da uno di questi tre valori:

- **Ob**: obbligatorio;
- **D**: desiderabile;
- **Op**: opzionale.

Ogni *requisito*_G viene poi associato al Caso d'Uso cui fa riferimento.

2.2.4 Codifica del software

L'attività di codifica del software viene svolta dai programmatori e rappresenta l'implementazione tramite codice del lavoro effettuato dai progettisti. Il software risultante dall'attività di codifica deve rispettare i requisiti individuati dagli analisti, soddisfare le metriche di verifica e *validazione*_G, e deve inoltre soddisfare le esigenze del proponente.

La codifica deve essere effettuata seguendo specifiche pratiche e diversi standard. Di seguito sono elencate le caratteristiche che il codice sviluppato dal team dovrà rispettare.

2.2.4.1 Stile della codifica

Per garantire uniformità, leggibilità e manutenibilità del codice, il team rispetterà le seguenti regole sintattiche e stilistiche:

- **Convenzioni di Nomenclatura:**

- Backend:
 - * **Classi**: Pascal Case;
 - * **Variabili e Metodi**: Snake Case;
 - * **Nomi file**: Snake Case;
 - * **Costanti**: Upper Snake Case.

- Frontend:
 - * **Classi**: Pascal Case;
 - * **Variabili e Metodi**: Camel Case;
 - * **Nomi file**: Kebab Case;
 - * **Costanti**: Upper Snake Case.

- **Formattazione:**

- **Indentazione**: si utilizzano 4 spazi per ogni livello di annidamento;
- **Lunghezza righe**: per favorire la lettura su diversi schermi, ogni riga di codice non deve superare i 100 caratteri;
- **Blocchi di codice**: le funzioni e i blocchi logici devono essere separati da una riga vuota per migliorarne la visibilità.

- **Best practices:**

- **Lunghezza metodi:** ogni metodo non dovrebbe superare le 20-30 righe di codice; se eccede, va rifattorizzato in sotto-metodi.
- **Annidamento:** è caldamente consigliato limitare i cicli e le condizioni annidate a un massimo di 3 livelli.
- **Variabili globali:** l'uso di variabili globali è da evitare dove possibile per prevenire effetti collaterali indesiderati.
- **Gestione errori:** le anomalie devono essere gestite in modo esplicito tramite l'uso di eccezioni, mantenendo il flusso di lavoro stabile.

- **Lingua e Commenti:**

- **Lingua:** tutta la codifica, inclusi nomi di variabili, metodi e commenti, deve essere rigorosamente in lingua inglese.
- **Commenti:** non devono descrivere il "come" (spiegato dal codice stesso), ma il "perché" di una determinata scelta implementativa.
- **Documentazione (Docstrings):** ogni funzione o metodo complesso deve essere preceduto da un commento descrittivo che ne specifichi lo scopo e gli argomenti.

2.2.4.2 Strumenti e Tecnologie

- **Tecnologie di sviluppo:**

- **Python:** utilizzato come linguaggio di programmazione ad alto livello orientato agli oggetti per la realizzazione della logica di backend. È stato scelto per la sua versatilità e l'ampia disponibilità di risorse.
- **React:** framework utilizzato per lo sviluppo dell'interfaccia grafica (frontend) dell'applicazione, permettendo un'interazione fluida e responsive.
- **FastAPI:** framework web moderno, veloce e ad alte prestazioni per la creazione di API RESTful con Python.

- **Infrastruttura:**

- **Docker:** piattaforma per eseguire l'applicazione in container, ovvero ambienti isolati che includono tutto il necessario per funzionare correttamente in qualsiasi ambiente di esecuzione. Garantisce la portabilità e la stabilità del sistema indipendentemente dalla macchina locale dello sviluppatore.

- **Strumenti di supporto e versionamento:**

- **Git:** Version Control System (VCS) utilizzato per tracciare l'evoluzione del codice sorgente e gestire gli sviluppi in branch separati.

- **GitHub:** Servizio di hosting per i *repository_G* *Git_G* che facilita la collaborazione asincrona tra i membri del team, il sistema di *ticketing_G* (Issue) e il monitoraggio delle Project Board.
 - **Visual Studio Code (VS Code):** IDE o *editor_G* di testo consigliato per la sua leggerezza e il supporto a estensioni personalizzabili che migliorano la produttività.
- *Automazione_G:*
 - **GitHub Actions:** Utilizzate per implementare la *Continuous Integration_G* (CI), automatizzando test e verifiche ad ogni *commit_G* o *pull_G* request per evitare l'introduzione di codice non funzionante nel *branch_G* principale.

3 Processi di supporto

I processi di supporto comprendono l'insieme di attività che forniscono assistenza ai processi primari e organizzativi durante l'intero ciclo di vita del prodotto software. Essi sono finalizzati a garantire la qualità del prodotto e del processo, a gestire in modo controllato le modifiche, la documentazione e le configurazioni, nonché a fornire i meccanismi necessari per la verifica, la *validazione* e la risoluzione dei problemi.

I processi di supporto che verranno descritti in questa sezione sono:

- **Documentazione**
- **Gestione delle configurazioni**
- **Accertamento della qualità**
- **Verifica**
- **Validazione**

3.1 Documentazione

La documentazione è uno *strumento* potente usato per definire, descrivere, tracciare e supportare le attività svolte durante il progetto. Nel dettaglio, il processo di documentazione supporta la tracciabilità delle informazioni, assicurandone la registrazione e la conservazione lungo tutte le fasi del ciclo di vita delle attività e dei processi.

3.1.1 Tipologie di documenti

Il gruppo ha creato dei **template** per ogni tipo di documento e ogni documento ha una struttura specifica. I tipi di documenti con un template sono:

- **Verbale interno**
- **Verbale esterno**
- **Diario di bordo**

3.1.1.1 Verbali

Generalmente i verbali seguono questa struttura:

- **Frontespizio**, composto da:
 - Logo e nome del gruppo
 - Tipologia del documento e le informazioni di riferimento, tra cui versione, data e luogo
 - La *lista* dei partecipanti alla riunione

- La **tavella delle versioni** che riporta le modifiche apportate al documento, con la data, il nome dell'autore, del *verificatore_G* e una breve descrizione del cambiamento;
- **Abstract** che ha lo scopo di indicare il periodo della riunione e i principali argomenti trattati;
- **Ordine del giorno** che indica in modo strutturato gli argomenti discussi durante la riunione;
- **Discussione**, dove vengono documentati i temi affrontati, le osservazioni dei partecipanti e le eventuali divergenze o punti di accordo;
- La sezione **Decisioni prese** documenta le decisioni prese dal gruppo, organizzate in tabella per facilitare la consultazione e il monitoraggio delle azioni conseguenti;
- La sezione **Attività da svolgere** elenca in modo dettagliato i compiti assegnati ai partecipanti, specificando i responsabili.

Ogni decisione ha un suo codice univoco e segue questo formato : "Dx (x numero della decisione)-ANNO/MESE/GIORNO_vx" ("vi" se interno, "ve" se esterno).

Anche le attività hanno codici univoci e seguono questo formato : "Ax (x numero dell'attività)-ANNO/MESE/GIORNO_vx" ("vi" se interno, "ve" se esterno).

3.1.1.2 Diario di bordo

Questi documenti, creati nell'ambito dell'attività coordinata dal Prof. Tullio Vardanega, sono utilizzati per monitorare lo stato di avanzamento di ciascun gruppo del I lotto e hanno carattere puramente informativo. Sono presentazioni semplici che trattano tre argomenti:

- Risultati raggiunti, qui vengono elencate le attività che sono state fatte nel periodo corrente;
- Prossime attività, un elenco di attività pianificate dal gruppo;
- Difficoltà riscontrate, eventuali difficoltà riscontrate durante lo svolgimento delle attività.

3.1.1.3 Denominazione dei documenti

I documenti, per la denominazione, seguono questo formato:

- **Verbali interni:** VI-ANNO-MESE-GIORNO
- **Verbali esterni:** VE-ANNO-MESE-GIORNO
- **Diari di bordo:** DB-ANNO-MESE-GIORNO

È stato deciso di adottare il formato **AAAA-MM-GG** per le date, al fine di semplificare la lettura e l'ordinamento cronologico dei documenti. La versione del documento si trova all'interno del file.

3.1.2 Strumenti utilizzati

Per compilare la documentazione il gruppo utilizza tre strumenti utili:

- **LaTeX**: un linguaggio mark-up usato per la redazione di documenti di alta qualità, specialmente scientifici e accademici. Richiede la compilazione a ogni nuova modifica, a differenza di altri linguaggi mark-up come ad *esempio* Typst. Il gruppo utilizza esclusivamente *LaTeX* per scrivere tutta la documentazione.
- **Overleaf**: è una piattaforma online che consente di scrivere, modificare e compilare documenti *LaTeX* direttamente dal browser, facilitando la collaborazione in tempo reale e la gestione dei progetti senza installare software locali. Il gruppo utilizza questo *editor* online per scrivere i documenti piccoli come i verbali.
- **GitHub**: l'adozione di *GitHub* consente di semplificare i processi di verifica e approvazione dei documenti, garantendo al contempo il controllo delle modifiche tramite le *Pull Request*. Permette inoltre di pianificare i passi successivi usando il sistema di Ticketing.

3.1.3 Attività previste

Il processo di documentazione prevede le seguenti attività:

1. **Produzione**: produzione del documento;
2. **Manutenzione**: insieme di attività da svolgere quando un documento deve essere modificato.

3.1.4 Produzione

La creazione di un documento prevede questi passaggi:

- **Creazione della issue su GitHub tramite sistemi di ticketing**: viene aperta una issue per poter tenere traccia dell'andamento e per assegnare il *responsabile* di quel documento. Per documenti grandi vengono creati dei *branch* secondari per poter effettuare modifiche senza andare a modificare il *branch* principale. I documenti piccoli come i verbali vanno fatti direttamente in Overleaf senza dover aprire *branch* secondari.
- **Inizio stesura**: La creazione del documento è determinata dall'assegnatario: ogni documento è prodotto dal soggetto *responsabile* delle relative attività.
- **Verifica**: dopo la stesura il file va verificato tramite *Pull Request* di *GitHub* oppure se è un verbale va verificato direttamente su Overleaf.

Dopo la verifica, il *responsabile* approva la *Pull Request* e il file viene integrato nel *branch* principale. Successivamente, il sistema automatizzato **GitHub Actions** provvede a inserire il file nella cartella corretta del sito web.

3.1.5 Manutenzione

È prevista la manutenzione della documentazione quando occorre apportare cambiamenti o aggiornamenti ai contenuti. Per fare una modifica si seguono i seguenti passi:

1. Creare il nuovo *branch_G* di lavoro;
2. Effettuare le modifiche in locale;
3. Dopo aver modificato il file, modificare il registro delle modifiche aggiungendo una nuova riga e fare il *push_G* sul *branch_G* di lavoro;
4. Aprire una *Pull_G* Request per consentire la verifica della qualità e della correttezza del documento;
5. Integrare il documento nel *branch_G* principale solo se il *responsabile_G* conferma la correttezza delle modifiche.

3.2 Gestione delle configurazioni

La gestione delle configurazioni è il processo che identifica le componenti software in un progetto e ne controlla le modifiche, i rilasci e gli stati di sviluppo.

3.2.1 Strumenti utilizzati

Per garantire lo svolgimento delle attività previste, il team utilizza:

- **GitHub:** per gestire creazione, modifiche e release di documenti e codice. Oltre a gestire il *versionamento_G*, *GitHub_G* permette anche l'assegnazione dei compiti e il loro tracciamento.

3.2.2 Attività previste

Il processo di gestione delle configurazioni comprende le seguenti attività:

1. **Identificazione della configurazione:** definizione univoca degli elementi software che compongono il progetto e delle modalità con cui essi vengono identificati e versionati nel tempo;
2. **Controllo della configurazione:** gestione delle modifiche agli elementi software, comprendente la loro identificazione, approvazione, implementazione, verifica e rilascio;
3. **Rendicontazione dello stato della configurazione:** raccolta, registrazione e mantenimento della storia degli elementi software e delle loro modifiche;
4. **Valutazione della configurazione:** verifica della completezza funzionale del software, intesa come soddisfacimento dei requisiti, e della completezza fisica degli elementi software rispetto alle *baseline* definite.

3.2.3 Identificazione della configurazione

L'attività di identificazione della configurazione riveste un ruolo essenziale nel progetto didattico. Essa prevede una descrizione accurata degli elementi software che compongono il sistema, svolta dai progettisti e formalizzata nel documento *Specifica tecnica*. Tale documento viene redatto successivamente al raggiungimento della *Requirements and Technology Baseline* e verrà reso disponibile in vista della definizione della *Product Baseline*.

3.2.4 Controllo della configurazione

Il team ha deciso di utilizzare i seguenti strumenti messi a disposizione dalla piattaforma *GitHub*_G per svolgere tale attività:

- Pull Request
- Issue
- Project Board

3.2.4.1 Pull Request

Le **Pull Request** sono richieste effettuate da un membro del team per proporre l'operazione di *merge*_G tra due *branch*_G differenti. Ogni *Pull*_G Request deve essere approvata dai verificatori prima di poter essere integrata nel *branch*_G principale, riducendo così la probabilità di introdurre errori nel codice o nella documentazione.

3.2.4.2 Issue

Ogni modifica a un componente software o a un documento viene tracciata mediante una **issue**. Ciascuna issue:

- è identificata su *GitHub*_G tramite la notazione **#NUMERO**, dove **NUMERO** rappresenta un identificativo univoco all'interno del *repository*_G;
- fa riferimento a una e una sola milestone di progetto e a una Project Board, entrambe appartenenti allo stesso *repository*_G della issue;
- è caratterizzata da un titolo e da una descrizione che ne specificano il contenuto;
- può avere uno o più assegnatari responsabili dello svolgimento dell'attività;
- viene chiusa quando il documento o il componente software a cui la issue fa riferimento viene integrato (*merge*_G) nel *branch*_G principale.

3.2.4.3 Project Board

La **Project Board** viene utilizzata per migliorare l'organizzazione del lavoro asincrono svolto dal team. Ogni issue creata viene inizialmente inserita nella sezione *Backlog* della Project Board, relativa allo sprint in corso. Quando un membro del team prende in carico una issue, questa viene spostata nella sezione *In progress*. Al termine dell'attività, e contestualmente

all'apertura della relativa *Pull_G* Request, la issue viene collocata nella sezione *In review*, in attesa della verifica da parte dei verificatori. Una volta approvata la *Pull_G* Request e completata l'operazione di *merge_G*, la issue viene spostata nella sezione *Done* e contrassegnata come chiusa.

3.2.5 Rendicontazione dello stato della configurazione

L'attività di rendicontazione dello stato della configurazione viene effettuata mediante il seguente sistema di *versionamento_G*: **MAJOR.MINOR.PATCH**, dove:

- la cifra **PATCH** viene incrementata in caso di modifiche di lieve entità al codice o alla documentazione, quali, ad *esempio_G*, la correzione di errori ortografici;
- la cifra **MINOR** viene incrementata a fronte di modifiche rilevanti al codice o alla documentazione;
- la cifra **MAJOR** viene incrementata quando il codice o la documentazione vengono approvati per una release ufficiale.

Tale sistema di *versionamento_G* è stato adottato dal team tramite la decisione [D2-2025/11/07_vii](#).

Per quanto riguarda la documentazione, ogni modifica viene inoltre segnalata all'inizio del documento mediante il registro delle modifiche.

3.2.6 Valutazione della configurazione

L'attività di valutazione della configurazione viene affrontata dal team mediante un'attività di tracciamento dei requisiti, definita all'interno del documento di [Analisi dei Requisiti](#). Tale attività viene svolta al fine di garantire che il prodotto software risulti funzionalmente completo, ossia che fornisca una risposta a tutti i requisiti richiesti dall'azienda proponente.

3.3 Accertamento della qualità

Il processo di accertamento della qualità ha lo scopo di fornire adeguate garanzie che il prodotto software, i processi di *sviluppo_G* e la documentazione associata siano conformi ai requisiti specificati, agli standard adottati e alle pianificazioni stabilite.

3.3.1 Attività previste

Il processo di accertamento della qualità comprende le seguenti attività:

1. **Implementazione del processo:** definizione, adozione e applicazione degli standard di qualità, delle metriche e delle procedure da utilizzare durante l'intero ciclo di vita del prodotto software.
2. **Accertamento del prodotto:** verifica della conformità del prodotto software e della documentazione associata ai requisiti specificati, agli standard di qualità definiti e ai criteri di accettazione stabiliti.

3. **Accertamento del processo:** verifica dell'aderenza dei processi del ciclo di vita del software alle procedure, agli standard e alle pianificazioni stabilite, valutandone la corretta applicazione e l'efficacia.

Le metriche individuate dal team di progetto sono descritte in modo dettagliato sia nel documento [Piano di Qualifica](#) sia nelle sezioni [6](#) e [7](#) del documento attuale. L'andamento delle attività del team di progetto viene monitorato e aggiornato al termine di ogni sprint e riportato nel Piano di Qualifica mediante grafici appositamente elaborati.

3.4 Verifica

Il processo di verifica ha lo scopo di determinare se il prodotto software sviluppato sia conforme ai requisiti specificati e alle condizioni imposte. Al fine di ridurre i costi e massimizzare l'[efficienzaG](#), tale processo viene integrato il più precocemente possibile all'interno del ciclo di vita del software, accompagnando le attività di sviluppo.

3.4.1 Strumenti utilizzati

Per garantire lo svolgimento delle attività previste, il team utilizza:

- **GitHub:** per la gestione del sistema di *Pull_G Request* (descritto nella sottosezione [3.2.4.1](#)), impiegato per supportare l'attività di verifica dei documenti.

3.4.2 Attività previste

Il processo di verifica comprende le seguenti attività:

1. **Implementazione del processo:** definizione e predisposizione del piano di verifica, nonché identificazione delle possibili criticità e dei rischi di non conformità nel progetto;
2. **Verifica:** esecuzione delle attività di verifica volte ad accertare la conformità di tutte le componenti di progetto (**processi, requisiti, progettazione, codice, integrazione e documentazione**) ai requisiti specificati, agli standard di riferimento e alle metriche e procedure definite dal team di progetto.

3.4.3 Implementazione del processo

L'attività di implementazione del processo si basa sull'individuazione delle criticità che possono manifestarsi nel corso del progetto didattico. Nella sezione 3 del documento [Piano di Progetto](#) è riportata un'accurata analisi dei rischi che include gran parte delle criticità previste, insieme alla probabilità di occorrenza e alle relative strategie di mitigazione. Tuttavia, in coerenza con quanto previsto dallo standard ISO/IEC 12207:1995 si riportano di seguito ulteriori criticità che potrebbero manifestarsi durante lo svolgimento del progetto didattico:

- **Disponibilità di fondi:** rischio di superamento del tetto di spesa massimo stabilito prima dell'aggiudicazione degli appalti. Tale rischio viene mitigato mediante un'accurata

pianificazione delle attività e dei ruoli all'interno degli sprint. Inoltre vengono svolte regolarmente riunioni interne per individuare tempestivamente ulteriori problemi;

- **Rischi associati alla codifica del prodotto software.** Questo rischio viene mitigato tramite un'accurata attività di verifica del software, descritta nella seguente sottosezione.

3.4.4 Verifica

L'attività di verifica della documentazione avviene tramite il sistema di *Pull_G Request*, descritto nelle sottosezioni [3.1.4](#) e [3.1.5](#).

Le modalità di verifica del codice, invece, verranno definite successivamente, a *Requirement and Technology Baseline* inoltrata, poiché sarà in quel momento che il team avrà necessità di approfondire tale argomento. In generale si distinguono due metodi principali per svolgere l'attività di verifica: **l'analisi statica** e **l'analisi dinamica**.

3.4.4.1 Analisi statica

L'analisi statica è definita tale in quanto non richiede l'esecuzione del programma oggetto di verifica. Essa mira ad assicurare la conformità alle regole e agli stili definiti, il rispetto delle proprietà attese e l'assenza di errori.

L'analisi statica può essere realizzata mediante due principali tecniche:

1. **Walkthrough:** si presuppone la presenza di difetti senza conoscerne a priori la posizione. Viene quindi effettuata un'analisi critica completa dell'intero oggetto di verifica. L'attività di Walkthrough viene svolta dagli sviluppatori e richiede un elevato impiego di tempo e risorse.
2. **Inspection:** anche in questo caso si presuppone la presenza di errori, ma l'analisi è mirata a specifiche parti dell'oggetto di verifica. Questa attività è svolta dai verificatori ed è più facilmente automatizzabile rispetto al metodo precedente.

Entrambe queste tecniche sono definite **di lettura**, in quanto si basano sull'analisi dell'oggetto di verifica tramite lettura, che può essere umana o automatizzata.

3.4.4.2 Analisi dinamica

L'analisi dinamica, al contrario dell'analisi statica, richiede l'esecuzione dell'oggetto da verificare. Essa studia il comportamento di singole parti del programma su un insieme finito di "casi prova" mediante dei **test**, che devono essere ripetibili e automatizzabili.

Ogni test è caratterizzato da:

- **Stato iniziale** dell'esecuzione;
- **Dati in ingresso:** dati che vengono inseriti nel programma;
- **Dati in uscita:** risultati attesi dall'esecuzione del programma, dato lo specifico input.

I test effettuati dal team, durante lo svolgimento del progetto didattico verranno identificati mediante il formato **T-Id-Tipologia**, dove:

- **T** sta per test;
- **Id** è un numero maggiore di zero, identificativo univoco di ogni test;
- **Tipologia** indica la tipologia del test. È rappresentata da uno di questi cinque valori:
 - **U**: unità;
 - **I**: integrazione;
 - **S**: sistema;
 - **R**: regressione;
 - **A**: accettazione.

Come si può notare dalle diverse tipologie, esistono vari tipi di test.

3.4.4.2.1 Test di unità

I test d'unità consistono nel testare singole unità software, quali funzioni, metodi o classi, in modo isolato rispetto al resto del sistema.

L'*obiettivo*_G principale dei test d'unità è verificare che ciascuna unità produca i risultati attesi a fronte di *input*_G noti, individuando eventuali errori logici o di implementazione nelle prime fasi dello sviluppo. Questi si dividono in due categorie:

- **Test funzionali:** detti anche **black-box** dato che fanno riferimento esclusivamente alla specifica degli ingressi e delle uscite dell'oggetto di verifica, senza considerarne la logica interna;
- **Test strutturali:** detti **white-box**, verificano la logica interna del codice dell'oggetto di verifica.

3.4.4.2.2 Test di integrazione

I test d'integrazione vengono eseguiti per verificare che le diverse unità software interagiscano correttamente fra loro.

Essi vengono sviluppati in modo incrementale, aumentando ad ogni passo le funzionalità disponibili del sistema. Si dividono in due categorie:

- **Bottom-up:** vengono sviluppate e integrate inizialmente le componenti con minori dipendenze d'uso e con maggiore utilità interna;
- **Top-down:** vengono sviluppate e integrate inizialmente le componenti con maggiori dipendenze d'uso e quindi maggiore valore aggiunto dal punto di vista esterno.

3.4.4.2.3 Test di sistema

I test di sistema verificano che il comportamento complessivo del sistema soddisfi tutti i requisiti software, definiti durante l'attività di analisi dei requisiti. Essi rientrano nella categoria dei test funzionali.

3.4.4.2.4 Test di regressione

I test di regressione vengono eseguiti per verificare che modifiche o aggiornamenti al software non abbiano introdotto nuovi difetti o problemi nelle funzionalità già esistenti.

3.4.4.2.5 Test di accettazione

I test di accettazione verificano che l'esecuzione del sistema soddisfi tutti i requisiti lato utente, definiti nel *capitolato_G* di presentazione.

3.5 Validazione

Il processo di *validazione_G* ha lo scopo di determinare se i requisiti definiti e il prodotto software finale soddisfino lo specifico uso inteso, così come concordato con gli utenti e gli stakeholder. In sintesi, il processo di *validazione_G* fornisce evidenza oggettiva della correttezza del prodotto dal punto di vista dell'utente finale, garantendo l'allineamento tra requisiti, implementazione e contesto operativo reale.

3.5.1 Attività previste

1. **Implementazione del processo:** definizione e predisposizione del piano di *validazione_G*, in accordo con gli standard utilizzati, al fine di stabilire criteri, metodi, responsabilità e risorse necessarie. Tale attività è finalizzata all'individuazione di eventuali non conformità rispetto ai requisiti e agli standard adottati, nonché alla pianificazione delle relative azioni correttive;
2. **Validazione:** esecuzione delle attività di *validazione_G* previste dal piano, attraverso test, revisioni e valutazioni del prodotto software nel suo contesto di utilizzo. I risultati ottenuti vengono documentati e analizzati per verificare la soddisfazione dei requisiti utente e per supportare il processo di accettazione del prodotto.

4 Processi Organizzativi

I processi organizzativi, pur non producendo direttamente il software, supportano quelli primari e di supporto, regolando risorse, responsabilità e comunicazione, e consentendo il monitoraggio del progetto e interventi correttivi in caso di scostamenti dagli obiettivi.

Con lo scopo di garantire qualità, *efficienza*_G e sostenibilità lungo l'intero ciclo di vita del software, tra i processi organizzativi si individuano:

- **Gestione dei Processi**
- **Gestione delle infrastrutture**
- **Miglioramento del processo**
- **Formazione del personale**

4.1 Gestione dei Processi

Come previsto dallo Standard ISO/IEC 12207:1995, il processo organizzativo di gestione dei processi ha l'*obiettivo*_G di individuare le attività generali che ciascun membro del team è tenuto a svolgere. Si tratta di un'attività fondamentale per assicurare che il progetto venga portato a termine in modo efficiente e nel rispetto degli standard di qualità definiti.

4.1.1 Attività previste

Le attività previste per questo processo sono:

- **Inizializzazione:** definizione delle risorse necessarie e dei requisiti delle attività;
- **Pianificazione:** *stima*_G di tempi, costi e assegnazione dei compiti nel *Piano di Progetto*;
- **Esecuzione e Controllo:** monitoraggio costante dell'avanzamento dei task rispetto alle scadenze;
- **Revisione e Valutazione:** verifica periodica dei risultati prodotti per assicurarne la conformità;
- **Chiusura.**

4.1.2 Ruoli e responsabilità

In conformità a quanto descritto nel regolamento del progetto didattico offerto dal committente, ogni componente del gruppo dovrà assumere ciascun ruolo per un numero di ore significativo. La gestione dei processi coinvolge tutti i membri del team, ciascuno secondo il proprio ruolo, e ognuno è *responsabile*_G dell'esecuzione delle attività assegnate, del rispetto delle scadenze e della segnalazione tempestiva di eventuali criticità o impedimenti. I ruoli richiesti dal progetto sono descritti qui di seguito.

4.1.2.1 Responsabile

È la figura centrale di coordinamento e il punto di riferimento primario per le comunicazioni con il committente e il proponente. Ha la responsabilità finale sulla pianificazione, la gestione delle risorse umane e il monitoraggio dell'avanzamento del progetto rispetto alle scadenze prefissate. Tra i suoi compiti principali figurano l'approvazione formale di tutta la documentazione, la gestione e mitigazione dei rischi e l'assegnazione delle issue ai membri del team.

4.1.2.2 Amministratore

L'Amministratore ha il compito fondamentale di gestire e mantenere l'infrastruttura tecnologica necessaria allo *sviluppo_G*, assicurandone l'*efficienza_G* e la sicurezza. È *responsabile_G* della gestione delle configurazioni, del *versionamento_G* del codice e della documentazione, nonché dell'implementazione di automazioni per ottimizzare il Way of Working. Supervisiona inoltre la corretta applicazione delle norme di progetto e la risoluzione di eventuali malfunzionamenti degli strumenti di supporto.

4.1.2.3 Analista

Questo ruolo è cruciale nelle fasi iniziali per trasformare i bisogni del proponente in requisiti tecnici dettagliati e non ambigui. L'Analista studia il dominio del problema, individua i casi d'uso e redige il documento di *Analisi dei Requisiti*, che funge da base solida per tutta la progettazione successiva. Deve assicurarsi che ogni *requisito_G* (funzionale, di qualità o di vincolo) sia chiaramente definito per evitare errori che comprometterebbero l'intero progetto.

4.1.2.4 Progettista

Al Progettista spetta il compito di trasformare i requisiti in una struttura architettonale solida e scalabile. Effettua le *scelte tecnologiche_G* relative a *framework_G* e librerie, definisce l'interazione tra i componenti del sistema e produce i diagrammi *UML_G* (come i diagrammi delle classi) necessari alla codifica. Il suo *obiettivo_G* è garantire un basso accoppiamento tra i moduli e un'alta manutenibilità del prodotto finale.

4.1.2.5 Programmatore

Il Programmatore è *responsabile_G* dello *sviluppo_G* attivo del software, traducendo le specifiche tecniche e architetturali in codice sorgente funzionante. Deve rispettare rigorosamente gli standard di Clean Code e le convenzioni linguistiche stabilite. Oltre alla scrittura del codice, ha il compito di realizzare i test di unità automatici e di collaborare alla redazione del *Manuale Utente* per l'utilizzatore finale.

4.1.2.6 Verificatore

È il garante della qualità e ha il compito trasversale di controllare che ogni prodotto (documento o codice) sia conforme alle *Norme di Progetto* e ai requisiti stabiliti. Esegue attività di analisi statica e analisi dinamica, segnala le non conformità, fornisce feedback per le correzioni e redige la parte relativa agli esiti dei test nel *Piano di Qualifica*.

4.1.3 Coordinamento e comunicazioni

Per assicurare una circolazione delle informazioni rapida ed efficiente, di seguito viene illustrato il modo in cui il team collabora internamente e si relaziona con gli Stakeholder. Il coordinamento tra i membri del team avviene tramite incontri periodici e momenti di aggiornamento informale, mentre le comunicazioni sono gestite attraverso i canali definiti, così da garantire la tracciabilità e la trasparenza delle decisioni. Ogni componente del team è tenuto a mantenere un flusso comunicativo continuo, segnalando tempestivamente ritardi, rischi o eventuali necessità di supporto.

4.1.3.1 Comunicazioni Interne

Le comunicazioni interne riguardano esclusivamente i membri del gruppo e si distinguono in due categorie:

- **Comunicazioni asincrone:** viene utilizzato Whatsapp per scambi rapidi, informali e risoluzione di dubbi minori che non richiedono una riunione.
- **Comunicazioni sincrone (Riunioni):** per le discussioni complesse, le sessioni di brainstorming e il coordinamento settimanale. Viene utilizzato *Discord*_G, che permette comunicazione vocale e condivisione dello schermo, facilitando il lavoro collaborativo.

4.1.3.2 Comunicazioni Esterne

Le comunicazioni esterne sono gestite dal *Responsabile*_G e riguardano il rapporto con proponente e committente. Si distinguono in tre categorie:

- **Incontri SAL (Stato Avanzamento Lavori):** viene utilizzato Google Meet per riunioni con l'azienda proponente, per presentare il lavoro svolto e ricevere feedback. La periodicità concordata è ogni due settimane, ma può subire variazioni in base al carico di lavoro da svolgere.
- **Comunicazioni formali:** per richieste ufficiali o invio di documenti, il *Responsabile*_G utilizza esclusivamente la posta elettronica del team, avente il seguente indirizzo: team9.atlas@gmail.com
- **Supporto rapido:** per comunicazioni veloci e dubbi di minore entità viene utilizzato un canale Telegram condiviso con i referenti aziendali.

4.1.3.3 Verbali

Ogni incontro sincrono, interno o esterno, deve essere documentato tramite un verbale redatto dal *Responsabile*_G e verificato dal *Verificatore*_G stabilito. I verbali esterni, una volta redatti e giunti alla versione stabile v1.0.0, vengono inviati al proponente per l'approvazione e la firma.

4.1.4 Gestione delle attività

L'*obiettivo*_G di questo processo è garantire un'organizzazione efficiente del lavoro durante il progetto. In questa sezione viene descritto il metodo di lavoro che permette al team di organizzare i compiti, monitorare l'avanzamento e garantire il rispetto delle scadenze.

4.1.4.1 Metodologia di lavoro

Il team adotta un approccio **Agile** allo *sviluppo* per garantire flessibilità, maggiore produttività e una comunicazione trasparente sia internamente che verso gli Stakeholder. Il lavoro viene organizzato in cicli iterativi definiti Sprint, solitamente di durata bisettimanale, che permettono una scomposizione modulare delle attività e una pianificazione accurata basata su obiettivi misurabili. Le principali pratiche adottate dal gruppo includono:

- **Pianificazione dello Sprint:** all'inizio di ogni ciclo, il team definisce gli obiettivi da raggiungere e predisponde le unità di lavoro (issue) necessarie per lo svolgimento delle attività individuate.
- **Continuous Integration (CI):** si promuove l'integrazione continua per favorire la collaborazione, garantendo una distribuzione efficace delle responsabilità e permettendo di risolvere le criticità in modo mirato e rapido.
- **Revisione e Retrospettiva:** al termine di ogni iterazione, il gruppo valuta il lavoro prodotto e analizza i processi adottati per identificare aree di miglioramento, ottimizzando costantemente il proprio Way of Working.
- **Adattabilità:** questo metodo iterativo assicura che il progetto possa adattarsi agilmente a cambiamenti nei requisiti o nelle priorità che potrebbero emergere durante il ciclo di vita del software.

4.1.4.2 Strumenti di Ticketing

Per la gestione operativa il team utilizza le *GitHub* Issues come Issue Tracking System. Ogni unità di lavoro è rappresentata da una issue che deve contenere informazioni precise:

- **Titolo e Descrizione:** sintesi del compito da svolgere.
- **Assegnatario:** il membro del team *responsabile* dello svolgimento.
- **Etichette (Labels):** per categorizzare la natura del task (es. bug, documentation, feature).
- **Project:** il *GitHub* Project a cui la issue è associata.
- **Milestone:** il traguardo o la *revisione* di progetto a cui il task è associato.

4.1.4.3 La Board Kanban

L'avanzamento delle attività è monitorato visivamente tramite una Project Board organizzata in colonne che rappresentano lo stato dei task:

- **Backlog:** attività pianificate ma non ancora iniziate.
- **In Progress:** compiti attualmente in fase di svolgimento.
- **In Review:** attività completate che attendono la verifica da parte di un membro terzo.
- **Done:** task verificati, approvati e integrati nel ramo principale.

4.1.4.4 Workflow di un Task

Il ciclo di vita di ogni attività segue una procedura rigorosa per garantire la qualità:

- **Creazione della Issue:** il *Responsabile_G* (o l'Amministratore) apre l'Issue e la assegna.
- **Creazione delle sotto-Issue:** l'assegnatario, dopo aver studiato il compito da svolgere, individua e crea sotto-issue per scomporre il problema in parti più piccole.
- **Sviluppo:** l'assegnatario sposta il task in "In Progress" e lavora sulle sotto-issue su un *branch_G* dedicato nel repository.
- **Pull Request:** al termine del lavoro, viene aperta una *Pull_G* Request (PR) collegata all'Issue principale, la issue passa allo stato "In review".
- **Verifica:** il *Verificatore_G* esamina la PR. Se l'esito è positivo, effettua il *merge_G*; in caso contrario, richiede modifiche tramite commenti, e la issue torna nello stato "In Progress" fino alla correzione.
- **Chiusura:** una volta effettuato il *merge_G*, la Issue passa automaticamente in "Done" e il *branch_G* di lavoro viene eliminato.

4.2 Gestione delle infrastrutture

La gestione delle infrastrutture ha l'*obiettivo_G* di definire tutti gli strumenti, gli ambienti e le risorse tecnologiche necessarie per garantire l'*efficienza_G* dei processi primari e di supporto. Questo processo permette al team di lavorare in modo stabile e sicuro, evitando interruzioni dovute a problemi tecnici o carenze di configurazione.

4.2.1 Attività previste

Il processo si articola in tre fasi principali:

- **Attuazione del processo:** identificazione e selezione degli strumenti più idonei per il progetto.
- **Realizzazione dell'infrastruttura:** configurazione operativa degli ambienti di *sviluppo_G*, dei *server_G* e dei canali di comunicazione.
- **Manutenzione dell'infrastruttura:** monitoraggio costante e aggiornamento degli strumenti per risolvere malfunzionamenti o adattarsi a nuove esigenze emerse durante gli sprint.

4.2.2 Strumenti di supporto

Il gruppo fa uso di un insieme di strumenti per rendere più agevole la collaborazione:

- **Sviluppo e Containerizzazione:** *Git*_G è impiegato come sistema di controllo di versione per monitorare e gestire le modifiche al codice. Docker viene utilizzato per creare ambienti isolati, garantendo che l'applicazione funzioni correttamente in modo indipendente dalla macchina locale.
- **Hosting e Collaborazione:** *GitHub*_G funge da piattaforma di hosting per i *repository*_G remoti e consente di organizzare e seguire il backlog tramite Issue e Project Board.
- **Comunicazione:** per le comunicazioni asincrone e rapide si adottano WhatsApp e Telegram, mentre le riunioni sincrone interne vengono svolte su Discord. Gli incontri sincroni con il proponente esterno sono invece condotti tramite Google Meet.
- **Pianificazione e Documentazione:** Google Calendar è utilizzato per la gestione di appuntamenti e scadenze, mentre *LaTeX*_G rappresenta lo standard per la redazione tipografica della documentazione tecnica.
- **Visualizzazione e monitoraggio:** è stato predisposto un file condiviso Google Fogli, articolato in più schede, che offre una vista sempre aggiornata sulle ore pianificate, quelle effettivamente utilizzate e quelle residue per ciascun ruolo e membro del team, oltre a un dettaglio suddiviso per ogni sprint. Ogni componente è tenuto ad aggiornare le ore realmente svolte durante lo sprint, registrando ogni ora produttiva. È inoltre presente una rappresentazione grafica dei dati riportati.

4.2.3 Gestione repository

Si adotta un modello ispirato a *GitHub*_G Flow e semplificato rispetto al *Git*_G Flow tradizionale, in modo da avere un workflow leggero basato sui seguenti principi:

- **Branching:** il *branch*_G *main* contiene esclusivamente codice stabile e verificato, e documenti definitivi che vengono esposti nel [sito web del team](#). Ogni nuova funzionalità o correzione di bug deve essere sviluppata in un *branch*_G dedicato creato a partire dal ramo di *sviluppo*_G chiamato *develop*, che deriva da *main*, contenente i nuovi documenti e il codice su cui il team sta lavorando.
- **Nomenclatura branch:** i nuovi rami devono seguire uno schema identificativo ben definito:
 - se il lavoro riguarda la stesura di un documento, il *branch*_G deve avere il nome *docs/*nome-documento;
 - se il lavoro riguarda la stesura di codice, il *branch*_G deve avere il nome *feature/*nome-feature;
 - se il lavoro riguarda la correzione di bug, il *branch*_G deve avere il nome *bugfix/*nome-bugfix.

- **Integrazione:** l'unione del codice nel ramo principale avviene esclusivamente tramite *Pull_G Request*. Queste devono essere revisionate e approvate da almeno un *Verificatore_G* prima del merge.
- **Automazione (CI/CD):** sono configurate delle *GitHub_G Actions* per automatizzare compiti ripetitivi, come la compilazione della documentazione *Latex_G* in *PDF_G* e la sua pubblicazione sul sito vetrina del progetto.

4.3 Miglioramento del processo

Il processo di miglioramento ha l'*obiettivo_G* di individuare punti deboli nell'organizzazione del lavoro e ottimizzare il Way of Working del team nel corso del progetto. Si tratta di un ciclo continuo che accompagna tutte le fasi di *sviluppo_G* e gestione, e consente di stabilire, misurare e migliorare gli standard operativi adottati durante il ciclo di vita del software.

4.3.1 Attività previste

Il processo si articola in tre attività principali:

- **Inizializzazione dei processi:** definizione e formalizzazione dei processi organizzativi, con relativa documentazione nelle *Norme di Progetto*.
- **Valutazione dei processi:** verifica periodica dell'*efficacia_G* dei processi tramite analisi di metriche, dati storici e risultati ottenuti.
- **Miglioramento dei processi:** individuazione delle aree critiche e implementazione di soluzioni correttive, aggiornando la documentazione ufficiale quando necessario.

4.3.2 Strumenti per il miglioramento

Lo *strumento_G* principale per identificare le opportunità di miglioramento è la Sprint Retrospective. Al termine di ogni iterazione, il gruppo analizza ciò che ha funzionato e ciò che deve essere cambiato, attuando azioni correttive immediate per evitare di ripetere errori passati.

4.4 Formazione del personale

Il processo di formazione ha lo scopo di garantire che ogni membro del team acquisisca le competenze tecniche e metodologiche necessarie per svolgere il proprio ruolo in modo efficace ed efficiente. L'*obiettivo_G* è mantenere il gruppo costantemente aggiornato e allineato sulle tecnologie scelte e sugli standard di qualità prefissati, favorendo al contempo la crescita professionale durante lo svolgimento del progetto. La formazione può riguardare tecnologie, metodologie di *sviluppo_G*, strumenti di collaborazione o aspetti organizzativi interni.

4.4.1 Attività previste

In conformità allo standard ISO/IEC 12207:1995, il gruppo articola il processo in tre attività principali:

- **Pianificazione della formazione:** analisi dei requisiti del progetto per individuare le lacune di competenza e definire le risorse necessarie.
- **Sviluppo di materiale per la formazione:** ricerca, selezione e organizzazione di guide, tutorial, manuali ufficiali o mini-progetti di prova da mettere a disposizione del team.
- **Implementazione del piano per la formazione:** esecuzione delle attività di studio e verifica dell'apprendimento attraverso l'applicazione pratica.

4.4.2 Metodologia di apprendimento

Il team adotta un approccio "learning by doing", basato sullo studio autonomo e asincrono integrato dalla realizzazione pratica del prodotto. Ogni membro è libero di personalizzare il proprio percorso di studi in base alle proprie esigenze e interessi, ma è incoraggiata la condivisione interna della conoscenza: i membri più esperti in una determinata tecnologia sono tenuti a supportare i colleghi tramite canali di comunicazione dedicati.

4.4.3 Piano di formazione e tempistiche di esecuzione

La formazione è un processo continuo e dinamico che accompagna l'intero ciclo di vita del progetto. Il gruppo si impegna a:

- sfruttare i momenti in cui vengono assegnati task di minore entità durante gli Sprint per dedicarsi allo studio;
- pianificare, se necessario, slot temporali dedicati esclusivamente all'approfondimento delle tecnologie necessarie come il linguaggio Python, la libreria React, la containerizzazione con Docker o l'uso di *Latex*^G per la documentazione.

Si predilige l'uso di risorse gratuite reperibili online, come documentazione ufficiale, video tutorial e materiale di riferimento condiviso.

5 Standard ISO/IEC 9126 per la qualità

Il team ha scelto di utilizzare lo standard ISO/IEC 9126 come riferimento per l'analisi e la valutazione della qualità del prodotto software. ISO/IEC 9126 è uno standard internazionale che definisce un modello per la valutazione della qualità del prodotto software. Rappresenta un insieme di linee guida e criteri di valutazione per gli attributi chiavi della qualità del software. Identifica sei caratteristiche principali:

- **Funzionalità**
- **Affidabilità**
- **Efficienza**
- **Usabilità**
- **Manutenibilità**
- **Portabilità**

5.1 Funzionalità

La funzionalità è la capacità di un prodotto software di fornire funzioni che soddisfano esigenze stabilite dal proponente. I suoi sotto-attributi sono:

- **Adeguatezza:** misura quanto le funzionalità del software rispondono correttamente ai bisogni dell'utente;
- **Accuratezza:** la capacità del prodotto software di fornire i risultati concordati o i precisi effetti richiesti;
- **Interoperabilità:** la capacità del prodotto software di interagire con altri sistemi;
- **Sicurezza:** la capacità del prodotto software di proteggere informazioni e dati negando in ogni modo che persone o sistemi non autorizzati possano accedervi o modificarli;
- **Conformità:** è la conformità del software agli standard, alle norme e alle specifiche funzionali pertinenti.

5.2 Affidabilità

L'affidabilità è la capacità del prodotto software di mantenere uno specificato livello di prestazioni quando usato in date condizioni per un dato periodo. Le sue sotto-caratteristiche sono:

- **Maturità:** la capacità del software di gestire in modo stabile le operazioni, cercando di evitare errori, malfunzionamenti e risultati scorretti;
- **Tolleranza agli errori:** la capacità di mantenere livelli predeterminati di prestazioni anche in presenza di malfunzionamenti o usi scorretti del prodotto;

- **Recuperabilità:** la capacità del software di ripristinare le prestazioni desiderate dopo che si è verificato un *errore_G*;
- **Aderenza:** la capacità di aderire a standard, regole e convenzioni inerenti all'affidabilità.

5.3 Efficienza

L'*efficienza_G* è la capacità di fornire appropriate prestazioni relativamente alla quantità di risorse usate. Le sue sotto-caratteristiche sono:

- **Comportamento rispetto al tempo:** la capacità di fornire adeguati tempi di risorse, elaborazione e velocità di attraversamento, sotto condizioni specifiche e determinate;
- **Utilizzo delle risorse:** la capacità di utilizzare le risorse in maniera adeguata;
- **Conformità:** la capacità di aderire a standard e specifiche sull'efficienza.

5.4 Usabilità

L'*usabilità_G* è la capacità del software di essere compreso, appreso e utilizzato dall'utente in condizioni specifiche. Le sue sotto-caratteristiche sono:

- **Comprensibilità:** esprime quanto è facile comprendere i concetti del prodotto;
- **Apprendibilità:** la capacità di ridurre l'impegno richiesto dall'utente per imparare ad usare la sua applicazione;
- **Operabilità:** la capacità del software di consentire agli utenti di operare e controllare il sistema senza difficoltà;
- **Attrattiva:** la capacità del software di essere piacevole per l'utente che ne fa uso;
- **Conformità:** la capacità del software di aderire a standard o convenzioni relativi all'usabilità.

5.5 Manutenibilità

La manutenibilità è la capacità del software di essere modificato, includendo correzioni, miglioramenti o adattamenti. Le sue sotto-caratteristiche sono:

- **Analizzabilità:** rappresenta il grado di facilità con cui il codice può essere analizzato per individuare errori o difetti;
- **Modificabilità :** la facilità con cui il software può essere modificato per aggiungere nuove funzionalità o per cambiare quelle esistenti;
- **Stabilità:** la capacità del software di evitare effetti inaspettati derivanti da modifiche errate;
- **Testabilità:** la capacità di essere facilmente testato per validare le modifiche apportate al software.

5.6 Portabilità

La portabilità è la capacità del software di essere trasportato da un *ambiente di lavoro*_G ad un altro. Le sue sotto-caratteristiche sono:

- **Adattabilità:** la capacità del software di adattarsi a diversi ambienti senza richiedere modifiche significative al codice sorgente o all'*architettura*_G;
- **Installabilità :** la facilità con cui il software può essere installato in un nuovo ambiente;
- **Conformità:** la capacità del software di aderire a convenzioni sulla portabilità;
- **Sostituibilità:** la capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti nello stesso ambiente.

6 Metriche per la qualità del processo

In questa sottosezione vengono esposte tutte le metriche utilizzate dal team per verificare la qualità del processo, ordinate per processo.

Le metriche sono identificate secondo il seguente formato: **MPCX**, dove:

- **MPC** indica che si tratta di una metrica riguardante la qualità del processo;
- **X** indica il numero con cui viene identificata la metrica.

6.1 Processi primari

6.1.1 Fornitura

Budget at Completion (BAC)	
Codice	/
Formula	$BAC = 12705\text{€}$
Descrizione	Il BAC rappresenta il budget totale pianificato per il completamento del progetto. Di fatto è il costo accettato all'aggiudicazione.

Tabella 6: Descrizione della metrica "Budget at Completion"

Earned Value (EV)	
Codice	MPC01
Formula	$EV = BAC * (\text{Percentuale completamento progetto})$
Descrizione	L' EV rappresenta il valore guadagnato, ovvero il valore previsto di quanto è stato effettivamente realizzato in un certo momento.

Tabella 7: Descrizione della metrica "Earned Value"

Planned Value (PV)	
Codice	MPC02
Formula	$PV = BAC * (\text{Percentuale pianificata completamento progetto})$
Descrizione	Il PV è il valore che il progetto dovrebbe avere in un dato momento nel tempo, come previsto in fase di pianificazione. Il calcolo viene fatto prima dell'inizio dei lavori, così da avere una base su cui tracciare i progressi.

Tabella 8: Descrizione della metrica "Planned Value"

Actual Cost (AC)	
Codice	MPC03
Formula	$AC = (\text{Ore impiegate}) * (\text{Costo per ora})$
Descrizione	L' AC rappresenta la somma spesa per il progetto in quel determinato momento.

Tabella 9: Descrizione della metrica "Actual Cost"

Cost Variance (CV)	
Codice	MPC04
Formula	$CV = EV - AC$
Descrizione	Il CV rappresenta lo scostamento economico tra il valore del lavoro effettivamente completato e il costo sostenuto per realizzarlo. Viene calcolato come la differenza tra il valore guadagnato (EV) e il costo effettivo (AC).

Tabella 10: Descrizione della metrica "Cost Variance"

Scheduled Variance (SV)	
Codice	MPC05
Formula	$SV = EV - PV$
Descrizione	L' SV rappresenta lo scostamento fra il lavoro effettivamente completato e quello pianificato. Esprime la differenza fra il valore guadagnato (EV) e il valore pianificato (PV).

Tabella 11: Descrizione della metrica "Scheduled Variance"

Cost Performance Index (CPI)	
Codice	MPC06
Formula	$CPI = \frac{EV}{AC}$
Descrizione	Il CPI misura l'efficienza dei costi del progetto. Esprime il rapporto fra il valore del lavoro completato (EV) e il budget utilizzato per raggiungere tale valore.

Tabella 12: Descrizione della metrica "Cost Performance Index"

Schedule Performance Index (SPI)	
Codice	MPC07
Formula	$\text{SPI} = \frac{\text{EV}}{\text{PV}}$
Descrizione	L' SPI misura l'efficienza con cui il progetto sta rispettando la pianificazione temporale. Esprime il rapporto tra il valore del lavoro completato (EV) e il valore del lavoro pianificato (PV).

Tabella 13: Descrizione della metrica "Schedule Performance Index"

Estimate at Completion (EAC)	
Codice	MPC08
Formula	$\text{EAC} = \frac{\text{BAC}}{\text{CPI}}$
Descrizione	L' EAC rappresenta il budget stimato per il completamento del progetto. Esprime il rapporto fra il budget totale prefissato per il progetto (BAC) e l'efficienza dei costi del progetto (CPI).

Tabella 14: Descrizione della metrica "Estimate at Completion"

Estimate to Complete (ETC)	
Codice	MPC09
Formula	$\text{ETC} = \text{EAC} - \text{AC}$
Descrizione	L' ETC rappresenta il costo residuo per il completamento del progetto. Esprime la differenza fra il budget stimato per il completamento del progetto (EAC) e il costo effettivamente sostenuto fino a quel momento.

Tabella 15: Descrizione della metrica "Estimate to Complete"

6.1.2 Sviluppo

Requirement Stability Index (RSI)	
Codice	MPC10
Formula	$\text{RSI} = \left[1 - \left(\frac{\text{RA} + \text{RM} + \text{RR}}{\text{RI}} \right) \right] * 100$
Descrizione	L' RSI specifica quanto i requisiti del progetto sono stabili nel tempo. Si misura rapportando i requisiti iniziali (RI) a quelli aggiunti (RA), quelli modificati (RM) e quelli rimossi (RR).

Tabella 16: Descrizione della metrica "Requirement Stability Index"

Structural Fan-In (SFIN)	
Codice	MPC11
Esempio	$\text{SFIN}(\text{Procedura}) = \text{Numero di procedure che chiama questa procedura}$
Descrizione	La metrica SFIN indica il grado di riutilizzo di un dato modulo o componente (file, metodo, classe) da altre componenti. Un valore alto indica che il modulo è spesso riutilizzato e quindi potenzialmente critico. Un valore basso indica, invece, un maggiore isolamento. Nella formula sopracitata viene riportato l'esempio di una procedura, tuttavia lo stesso ragionamento può essere utilizzato per altre componenti del prodotto software.

Tabella 17: Descrizione della metrica "Structural Fan-In"

Structural Fan-Out(SFOUT)	
Codice	MPC12
Esempio	$\text{SFOUT}(\text{Procedura}) = \text{Numero di procedure chiamate da questa procedura}$
Descrizione	La metrica SFOUT indica il grado di accoppiamento di un dato modulo o componente (file, metodo, classe) ad altre componenti. Un valore alto indica un alto grado di accoppiamento che può rendere più difficoltoso il testing e la sua manutenzione. Un valore basso, al contrario, indica maggiore semplicità e coesione. Nella formula sopracitata viene riportato l'esempio di una procedura, tuttavia lo stesso ragionamento può essere utilizzato per altre componenti del prodotto software.

Tabella 18: Descrizione della metrica "Structural Fan-Out"

6.2 Processi di supporto

6.2.1 Documentazione

Indice di Gulpease	
Codice	MPC13
Formula	$\text{Gulpease} = 89 - \left(\frac{\text{Numero di lettere}}{\text{Numero di parole}} \right) * 10 + \left(\frac{\text{Numero di frasi}}{\text{Numero di parole}} \right) * 300$
Descrizione	L' indice di Gulpease è un indice di leggibilità del testo. Si misura rapportando il numero di parole di un testo al numero delle lettere e delle frasi. Un indice ≥ 80 indica che il testo è comprensibile anche per le persone che hanno completato solo la scuola elementare. Un indice compreso fra 60 e 80 indica che il testo è di difficoltà media, comprensibile da persona con la licenza media. Un indice di Gulpease compreso fra 40 e 60 indica che il testo è abbastanza difficile, comprensibile per chi ha almeno un diploma di una scuola superiore. Un indice < 40 indica che il testo è molto difficile, comprensibile dai lettori con un'istruzione universitaria.

Tabella 19: Descrizione della metrica "Indice di Gulpease"

Indice di correttezza ortografica	
Codice	MPC14
Descrizione	L' indice di correttezza ortografica misura il numero di errori ortografici presenti in un documento.

Tabella 20: Descrizione della metrica "Indice di correttezza ortografica"

6.2.2 Verifica

Code Coverage	
Codice	MPC15
Formula	$\text{Code Coverage} = \left(\frac{\text{Numero di linee di codice testate}}{\text{Numero di linee di codice totali}} \right) * 100$
Descrizione	Il Code Coverage rappresenta la percentuale del codice coperto da test automatizzati.

Tabella 21: Descrizione della metrica "Code Coverage"

Test Success Rate	
Codice	MPC16
Formula	$\text{Test Success Rate} = \left(\frac{\text{Numero di test superati}}{\text{Numero di test totali}} \right) * 100$
Descrizione	Il Test Success Rate rappresenta la percentuale dei test che vengono passati con esito positivo.

Tabella 22: Descrizione della metrica "Test Success Rate"

6.2.3 Qualità

Quality Metrics Satisfied	
Codice	MPC17
Formula	$\text{Quality Metrics Satisfied} = \left(\frac{\text{Numero di metriche soddisfatte}}{\text{Numero di metriche totali}} \right) * 100$
Descrizione	Il Quality Metrics Satisfied rappresenta la percentuale delle metriche di qualità soddisfatte.

Tabella 23: Descrizione della metrica "Quality Metrics Satisfied"

6.3 Processi organizzativi

6.3.1 Gestione dei processi

Non-Calculated Risk	
Codice	MPC18
Descrizione	L'indice Non-Calculated Risk rappresenta il numero di rischi non previsti che si sono manifestati durante lo svolgimento del progetto.

Tabella 24: Descrizione della metrica "Non-Calculated Risk"

7 Metriche per la qualità del prodotto

In questa sottosezione vengono esposte tutte le metriche utilizzate dal team per verificare la qualità del prodotto. Le metriche sono raggruppate in sottosezioni corrispondenti alle caratteristiche per la qualità dello standard ISO/IEC 9126, meglio precise nella sezione 5.

Le metriche sono identificate secondo il seguente formato: **MPDX**, dove:

- **MPD** indica che si tratta di una metrica riguardante la qualità del prodotto;
- **X** indica il numero con cui viene identificata la metrica.

7.1 Funzionalità

Requisiti Obbligatori Soddisfatti	
Codice	MPD01
Formula	Requisiti Obbligatori Soddisfatti = $\frac{\text{Requisiti obbligatori soddisfatti}}{\text{Requisiti obbligatori totali}} * 100$
Descrizione	La metrica Requisiti Obbligatori Soddisfatti rappresenta la percentuale dei requisiti obbligatori soddisfatti sul totale dei requisiti obbligatori individuati.

Tabella 25: Descrizione della metrica "Requisiti Obbligatori Soddisfatti"

Requisiti Desiderabili Soddisfatti	
Codice	MPD02
Formula	Requisiti Desiderabili Soddisfatti = $\frac{\text{Requisiti desiderabili soddisfatti}}{\text{Requisiti desiderabili totali}} * 100$
Descrizione	La metrica Requisiti Desiderabili Soddisfatti rappresenta la percentuale dei requisiti desiderabili soddisfatti sul totale dei requisiti desiderabili individuati.

Tabella 26: Descrizione della metrica "Requisiti Desiderabili Soddisfatti"

Requisiti Opzionali Soddisfatti	
Codice	MPD03
Formula	Requisiti Opzionali Soddisfatti = $\frac{\text{Requisiti opzionali soddisfatti}}{\text{Requisiti opzionali totali}} * 100$
Descrizione	La metrica Requisiti Opzionali Soddisfatti rappresenta la percentuale dei requisiti opzionali soddisfatti sul totale dei requisiti opzionali individuati.

Tabella 27: Descrizione della metrica "Requisiti Opzionali Soddisfatti"

7.2 Affidabilità

Branch Coverage	
Codice	MPD04
Formula	Branch Coverage = $\frac{\text{Branch eseguiti}}{\text{Branch totali}} * 100$
Descrizione	La metrica Branch Coverage rappresenta la percentuale di <i>branch</i> (rami) che sono stati eseguiti almeno una volta con l'esito atteso, rispetto al totale di <i>branch</i> presenti.

Tabella 28: Descrizione della metrica "Branch Coverage"

Statement Coverage	
Codice	MPD05
Formula	Statement Coverage = $\frac{\text{Statement eseguiti}}{\text{Statement totali}} * 100$
Descrizione	La metrica Statement Coverage rappresenta la percentuale di <i>statement</i> (istruzioni) eseguiti almeno una volta con l'esito atteso, rispetto al totale di <i>statement</i> presenti.

Tabella 29: Descrizione della metrica "Statement Coverage"

Failure Density	
Codice	MPD06
Formula	Failure Density = $\frac{\text{Numero di fallimenti totali}}{\text{Dimensione dell'Unità di codice}}$
Descrizione	La metrica Failure Density rappresenta il numero di fallimenti correttamente riscontrati durante i test per unità di dimensione del codice.

Tabella 30: Descrizione della metrica "Failure Density"

7.3 Usabilità

Time on Task	
Codice	MPD07
Formula	Time on Task = Tempo medio per completare un'attività
Descrizione	La metrica Time on Task rappresenta il tempo medio necessario per completare un'attività da parte dell'utente.

Tabella 31: Descrizione della metrica "Time on Task"

Error Rate	
Codice	MPD08
Formula	Error Rate = $\frac{\text{Numero di errori totali}}{\text{Numero di azioni totali}} * 100$
Descrizione	La metrica Error Rate rappresenta la percentuale di errori rispetto al numero totale di azioni eseguite.

Tabella 32: Descrizione della metrica "Error Rate"

7.4 Efficienza

Response Time	
Codice	MPD09
Formula	Response Time = Tempo medio di risposta
Descrizione	La metrica Response Time rappresenta il tempo medio impiegato dal sistema per rispondere a una richiesta.

Tabella 33: Descrizione della metrica "Response Time"

7.5 Manutenibilità

Code Smells	
Codice	MPD10
Formula	Code Smells = $\frac{\text{Numero di code smells}}{\text{Dimensione dell'Unità di codice}}$
Descrizione	La metrica Code Smells rappresenta il numero di <i>code smells</i> presenti in un'Unità di codice.

Tabella 34: Descrizione della metrica "Code Smells"

Coefficient of Coupling	
Codice	MPD11
Formula	$\text{Coefficient of Coupling} = \frac{\text{Numero di dipendenze}}{\text{Numero di componenti}}$
Descrizione	La metrica Coefficient of Coupling rappresenta il numero di dipendenze tra i componenti del sistema.

Tabella 35: Descrizione della metrica "Coefficient of Coupling"

Cyclomatic complexity	
Codice	MPD12
Formula	$\text{Cyclomatic complexity} = E - N + P$
Descrizione	E rappresenta il numero di archi nel grafo di controllo del programma, N rappresenta il numero di nodi nel grafo, e P rappresenta il numero di componenti connesse. La metrica Cyclomatic complexity quantifica la complessità logica e strutturale del programma contando il numero di percorsi linearmente indipendenti attraverso il suo codice sorgente.

Tabella 36: Descrizione della metrica "Cyclomatic complexity"

Parametri per metodo	
Codice	MPD13
Formula	$\text{Parametri}(\text{Metodo}) = \text{Numero di argomenti nella firma della funzione}$
Descrizione	La metrica Parametri per metodo rappresenta il numero di argomenti presenti nella firma di ciascun metodo all'interno del progetto.

Tabella 37: Descrizione della metrica "Parametri per metodo"

Linee di codice per metodo	
Codice	MPD14
Formula	$\text{Linee di codice}(\text{Metodo}) = \text{Linee di istruzioni nel metodo}$
Descrizione	La metrica Linee di codice per metodo rappresenta una misura quantitativa della dimensione di ciascun metodo all'interno del progetto.

Tabella 38: Descrizione della metrica "Linee di codice per metodo"

Profondità delle gerarchie	
Codice	MPD15
Formula	Profondità delle gerarchie = Massima profondità delle gerarchie di classi
Descrizione	La metrica Profondità delle gerarchie descrive l'equilibrio tra riutilizzo del codice e complessità strutturale. Rappresenta il livello massimo di profondità delle gerarchie di classi all'interno del progetto.

Tabella 39: Descrizione della metrica "Profondità delle gerarchie"