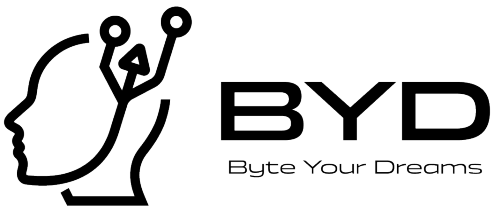

byteyourdreams.swe@gmail.com

Norme di progetto

Informazioni documento

***Redattore*$_G$**	A.M. Margarit
***Verificatore*$_G$**	A. Mio
Amministratore	L. Albertin
***Destinatari*$_G$**	T. Vardanega R. Cardin

Registro delle modifiche

***Versione*$_G$**	***Data*$_G$**	***Autore*$_G$**	***Verificatore*$_G$**	***Dettaglio*$_G$**
1.0.2	26/05/2025	Y. Huang	Y. Huang	Fix: update changelog to include version 1.0.1 in Norme di progetto document
1.0.1	26/05/2025	Y. Huang	A. Mio	Fix documento
1.0.0	09/02/2025	L.Albertin	L. Albertin	Approvazione documento
0.6.2	07/02/2025	L. Albertin	L. Zanesco	Fix documento
0.6.1	14/01/2025	L. Albertin	L. Zanesco	Fix documento
0.6.0	14/01/2025	A.M Margarit	L. Zanesco	Redazione sez. *Standard*$_G$ per la qualità
0.5.1	09/01/2025	L. Albertin	L. Zanesco	Fix documento
0.5.0	09/01/2025	A.M Margarit	L. Zanesco	Redazione sez. *Processi*$_G$ organizzativi e Gestione della qualità
0.4.3	04/01/2025	A.M Margarit	A. Mio	Fix documento
0.5.0	05/12/2024	A. Mio	O.F. Stiglet	Redazione sez. Fornitura e Sviluppo
0.4.2	01/12/2024	L.Albertin	O.F. Stiglet	Fix documento
0.4.1	16/11/2024	L.Albertin	O.F. Stiglet	Fix introduzione
0.4.0	16/11/2024	L.Albertin	O.F. Stiglet	Redazione sez. Verifica
0.3.0	15/11/2024	L.Albertin	Y. Huang	Redazione sez. Gestione della configurazione
0.2.0	13/11/2024	L.Albertin	Y. Huang	Redazione sez. Documen-tazione
0.1.0	12/11/2024	A. Mio	L. Albertin	Redazione sez. Fornitura
0.0.1	10/11/2024	A.M. Margarit	A. Mio	Redazione

Contents

1 Introduzione

1.1 Scopo del documento

Lo scopo principale del seguente documento è la definizione delle linee guida che ogni componente del *t*eam deve rispettare per assicurare che il processo di realizzazione del progetto didattico risulti efficiente ed efficace. Per definire i processi e le relative attività del seguente documento si è fatto riferimento allo *Standard*$_G$ **ISO/IEC 12207-1995**.

Il documento è organizzato secondo i processi del ciclo di vita del software e ogni processo si configura come una serie di attività, la quale è strutturata attraverso una gerarchia.

E' fondamentale evidenziare che questo documento è in continuo perfezionamento in quanto le norme intrinseche stabilite sono periodicamente rielaborate e ottimizzate.

1.2 *Glossario*$_G$

La documentazione contiene riferimenti al ***Glossario***, all'interno del quale vengono esposti i significati di tutti i termini, che potrebbero risultare troppo specifici o ambigui, pervenuti nei documenti relativi al progetto.
Quando un termine è scritto con una nota a pedice con la lettera *G* indica la possibilità di recuperare la definizione nel ***Glossario***.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- ***Standard*$_G$ ISO/IEC 12207-1995**:https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  Standard$_G$ internazionale che definisce un modello di ciclo di vita del software e in cui sono definite delle linee guida per la gestione dei processi software e le relative attività.

  Stabilisce una struttura per organizzare le diverse fasi e le attività coinvolte nel ciclo di vita del software, aiutando le organizzazioni a sviluppare prodotti software in modo più efficiente ed efficace.
- **Capitolato C2**: https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C2.pdf

1.3.2 Riferimenti informativi

- ***Glossario***;

2 *Processi*$_G$ primari

2.1 Fornitura

2.1.1 Introduzione

Il processo di fornitura di un software comprende tutte le attività necessarie per consegnare un prodotto software funzionante e conforme ai requisiti del committente. Si occupa di controllare e gestire le attività svolte dal team, per assicurare la copertura di tutti i requisiti e la consegna entro tempi e budget stabiliti. Il processo di fornitura viene avviato solo dopo aver completato la stesura del documento di Valutazione Capitolati. A seguito viene avviato stipulato un contratto con l'azienda proponente, dove saranno definiti i requisiti e i vincoli per la consegna del prodotto.

Nel resto del capitolo verranno analizzate nel dettaglio le attività coinvolte, le modalità di comunicazione con l'azienda proponente, la documentazione necessaria e gli strumenti utilizzati per gestire efficacemente il processo di fornitura del software.

2.1.2 attività

Le attività coinvolte nella fornitura, secondo quanto stabilito dallo standard ISO/IEC 12207:1995, del software sono molteplici e comprendono diverse fasi fondamentali:

1. **Definizione e raccolta dei requisiti**: Identificazione dei bisogni dell'azienda proponente e definizione dei requisiti funzionali e non funzionali;
2. **Pianificazione del progetto**: Elaborazione di un piano dettagliato con attività, tempistiche e risorse necessarie;
3. **Contrattazione**: Analisi di documenti tecnici, specifiche o capitolati.
4. **Pianificazione**: Vengono definiti i criteri di accettazione, metriche di qualità e strategie di testing.
5. **Esecuzione e controllo**: Progettazione, implementazione e versionamento del codice.;
6. **test e Verifica**: test unitari, di integrazione, di sistema e di accettazione;
7. **Comunicazioni tra cliente e gli stakeholder**: Riunioni periodiche, aggiornamenti SAL_G e risoluzione di problemi;
8. ***Deployment*$_G$ e consegna**: Installazione, configurazione del software nell'ambiente di produzione;

Un'efficace gestione di queste attività è essenziale per garantire una fornitura del software di successo, rispettando le tempistiche, il budget e gli standard di qualità richiesti.

2.1.3 Comunicazioni con l'azienda proponente e documentazione fornita

La comunicazione tra Byte Your Dreams e la proponente **Vimar S.p.A.** sarà un elemento cruciale che permetterà al gruppo di ottenere feedback su quanto fatto fino a quel momento. Una comunicazione chiara e strutturata consente di ridurre il rischio di malintesi e garantire la correttezza del prodotto. La comunicazione dovrà essere chiara e strutturata ed avverà tramite email per comunicazioni formali o tramite chat di Microsoft teams per comunicazioni veloci. Ogni 2 settimane, inoltre, sarà organizzato un incontro in cui si discuterà l'avanzamento del progetto e gli obbiettivi per il periodo successivo. Ad ogni incontro verrà redatto un verbale esterno, che includerà la data e l'ora dell'incontro, i partecipanti, gli argomenti trattati ed eventuali chiarimenti. I verbali esterni verranno archiviati nella directory "Documenti Esterni/Verbali" all'interno della repository "Documents" del gruppo Byte Your Dreams.

2.1.4 Documentazione fornita

Di seguito vengono elencati i documenti che verranno forniti ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin, e all'azienda proponente **Vimar S.p.A**.

2.1.4.1 Valutazione dei capitolati

Il documento "Valutazione dei Capitolati" contiene l'analisi dettagliata dei capitolati proposti, con la scelta effettuata dal gruppo Byte Your Dreams. Di seguito sono riportate le sezioni del documento:

- **Informativa sulla proponente**: nome azienda, nome progetto, informazioni utili;
- **Obiettivo**: fornisce una sintesi dell'aspettativa del prodotto software descritto nel capitolato.
- **Tecnologie suggerite**: include suggerimenti e consigli dalla proponente per lo sviluppo del prodotto.
- **Considerazioni**: Include le considerazioni del gruppo Byte Your Dreams, i pro, i contro, le criticità e le motivazioni che hanno influenzato la decisione di accettare o meno il capitolato, eventuali proposte di integrazione o modifica.

2.1.4.2 Analisi dei requisiti v1.0.0

L'analisi dei requisiti fornisce le basi per la progettazione, lo sviluppo e la validazione del sistema. Il documento "Analisi dei requisiti" fornisce la descrizione dei requisiti del capitolato, dei casi d'uso e delle funzionalità del prodotto finale. Lo scopo di questo documento è definire in modo chiaro e dettagliato i requisiti del capitolato, i casi d'uso e le funzionalità del prodotto. Eliminando così eventuali ambiguità e incomprensioni che potrebbero compromettere la qualità del prodotto finale.
Il documento comprende le seguenti sezioni:

- **Introduzione**: include una breve descrizione del prodotto e delle sue funzionalità.
- **Casi d'uso**: identifica tutti i possibili scenari di utilizzo del prodotto da parte dell'utente. Ogni caso d'uso è accompagnato da una descrizione che ne illustra il funzionamento.
- **Requisiti**: l'insieme di tutte le richieste e i vincoli definiti dalla proponente, o derivanti da discussioni con il gruppo, per la realizzazione del prodotto softwareG commissionato. Ogni requisito avrà una propria descrizione e il relativo caso d'uso.

2.1.4.3 Piano di progetto v1.0.0

Il piano di progetto descrive in dettaglio come il progetto sarà realizzato, gestito e controllato, fornendo una roadmap che guida l'intero ciclo di vita del progetto, dalla pianificazione iniziale alla consegna finale del software.
Il documento comprende le seguenti sezioni:

- **Analisi dei rischi**: contiene l'analisi dei rischi che il gruppo prevede di incontrare durante lo svolgimento del progetto. Per la prevenzione dei rischi e/o la loro eventuale mitigazione verrà inserito per ogni possibile rischio una descrizione, una procedura di identificazione, la probabilità di occorrenza, l'indice di gravità e un relativo piano di mitigazione.
- **Pianificazione e metodo utilizzato**: include una serie di accorgimenti per pianificare lo sviluppo delle attività, come: la suddivisione dei ruoli con relativa definizione delle attività specifiche per ciascun ruolo. Presenta una descrizione dettagliata del modello di sviluppo scelto, insieme alle motivazioni che hanno guidato la decisione verso tale approccio.
- **Preventivo e consuntivo**: contiene una stima delle ore e dei costi previsti per ciascuna attività, suddivisi per ogni periodo stabilito. Alla conclusione di ogni periodo, viene redatto un consuntivo riportante ore e costi sostenuti per ciascuna attività.

2.1.4.4 Piano di qualifica v1.0.0

Il piano di qualifica definisce gli approcci, le metodologie, gli strumenti e le risorse necessari per garantire che il software sviluppato soddisfi i requisiti di qualità stabiliti, rispettando gli standard e le aspettative dei committenti e degli stakeholder.
Il documento contiene le seguenti sezioni:

- **Qualità di processo**: include le strategie di verifica e validazione per garantire che i processi di sviluppo del prodotto raggiungano gli obiettivi di qualità.
- **Qualità di prodotto**: include le strategie di verifica e validazione per assicurare che il prodotto soddisfi gli obiettivi di qualità.
- **Specifica dei test**: include una descrizione dettagliata dei test.

2.1.4.5 *Glossario$_G$* v1.0.0

Il glossario offre una definizione chiara e univoca dei termini, concetti e abbreviazioni utilizzati all'interno della documentazione. La presenza di un glossario garantisce ai lettori una comprensione coerente dei concetti specifici presenti. Ogni termine deve essere descritto in modo semplice e preciso, in modo da ridurre eventuali ambiguità e prevenire malintesi.

2.1.4.6 Lettera di presentazione

La lettera di presentazione accompagna la consegna della documentazione e del prodotto software durante le fasi di revisione del progetto. Questo documento ha lo scopo di elencare dettagliatamente la documentazione prodotta, che sarà consegnata ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin, e all'azienda proponente **Vimar S.p.A.**.
Nella repository "Documents" è possibile consultare le seguenti lettere di presentazione: • Lettera di presentazione Candidatura • Lettera di presentazione *RTB*$_G$

2.1.5 Strumenti

L'adozione di strumenti adeguati è fondamentale per garantire la qualità, l'efficienza e la gestione ottimale di tutte le fasi del progetto. Gli strumenti sono utilizzati in ogni fase del ciclo di vita del software, dalla progettazione alla codifica, dai test alla gestione del progetto, e contribuiscono a migliorare la produttività, a ridurre il rischio di errori e a facilitare la collaborazione tra i membri del team. Un'accurata selezione degli strumenti giusti è essenziale per affrontare le sfide tecniche e operative che possono sorgere durante lo sviluppo del software. Ecco una panoramica delle principali categorie di strumenti utilizzati durante il processo di sviluppo software:

- **Microsoft teams**: Un servizio di videoconferenza per gli incontri e una piattaforma di messaggistica istantanea per le comunicazioni con la proponente;
- ***Git*$_G$ Flow**: Strategia di branching adottata dal team che facilita la gestione delle release e lo sviluppo collaborativo tramite l'utilizzo workflow strutturati
- **GitHub**: Piattaforme online che offrono repository *Git*$_G$ collaborative.
- ***Visual Studio Code*$_G$**: Editor di codice che supporta diversi linguaggi di programmazione.

2.2 Sviluppo

2.2.1 Introduzione

Il modello ISO/IEC 12207:1995 fornisce una linea guida per i processo di sviluppo del software, definendo i processi e le attività necessarie per produrre software di alta qualità. La sua adozione consente che ogni fase del processo di sviluppo sia affrontata in modo sistematico, con una chiara separazione delle responsabilità e delle attività, migliorando così la qualità e l'affidabilità del software. Questo modello si articola in diverse fasi, ognuna con i propri obiettivi e processi. Questo ne garantisce una corretta esecuzione e il rispetto delle normative e degli standard richiesti.

2.2.2 Analisi dei requisiti

2.2.2.1 Descrizione

L'analisi dei requisiti è un'attività fondamentale nello sviluppo di software: costituisce il punto di partenza per il design, l'implementazione e i test del sistema. Durante questa fase vengono raccolte, documentate, comprese e definite in maniera completa le esigenze del utente e del sistema. L'obiettivo principale dell'analisi dei requisiti è quello di assicurarsi che tutte le parti interessate abbiano una visione comune delle funzionalità e delle caratteristiche che il sistema deve possedere. Per fare ciò è necessario comprendere il dominio, ovvero conoscere i processi, le entità, le regole e le interazioni che caratterizzano quell'area specifica in cui il sistema sarà applicato. Durante la descrizione dei requisiti, è importante adottare un linguaggio preciso e privo di ambiguità. Ogni requisito deve essere scritto in modo che possa essere facilmente verificato e testato in seguito, attraverso la definizione di criteri di accettazione chiari. È fondamentale che il processo di descrizione dei requisiti sia iterativo e coinvolga costantemente gli stakeholder. Poiché i requisiti possono evolvere durante il ciclo di vita del software, la revisione e l'aggiornamento del documento dei requisiti è una pratica essenziale per garantire che il prodotto finale soddisfi le reali esigenze degli utenti.

2.2.2.2 Obiettivi

Gli obiettivi dell'analisi dei requisiti sono strettamente legati alla capacità di tradurre le necessità degli stakeholder in requisiti chiari, misurabili e realizzabili. In questa fase, si definiscono le linee guida che orienteranno tutte le fasi successive del progetto, dalla progettazione alla codifica. Gli obiettivi devono essere specifici, misurabili, raggiungibili, realistici e tempestivi, in modo che il team di sviluppo possa monitorare il progresso e garantire che il software finale risponda efficacemente alle esigenze degli utenti. Gli obiettivi si suddividono in diverse categorie, ognuna delle quali ha un impatto sul successo del progetto.

- Fornire ai Progettisti una base chiara e comprensibile per la definizione dell'architettura e il design del sistema;
- Fornire una base per la pianificazione mediante i requisiti raccolti;
- Facilitare la comunicazione tra fornitori e proponente;
- Fornire riferimenti per la verifica;
- Stabilire priorità;
- Verificabilità: creazione di requisiti verificabili in modo che ogni requisito debba essere definito in modo tale da poter essere testato durante le fasi successive del progetto.

2.2.2.3 Casi d'uso

I casi d'uso rappresentano uno degli strumenti più efficaci per descrivere il comportamento del sistema in modo chiaro e comprensibile. Forniscono una descrizione dettagliata delle funzionalità del sistema dal punto di vista degli utenti, identificando, per ogni caso d'uso, una sequenza di azioni che il sistema esegue in risposta a un'interazione dell'utente o di un sistema esterno. La creazione di casi d'uso ha come obiettivo la definizione delle funzionalità del software dal punto di vista dell'utente finale, favorendo la comprensione delle specifiche attraverso l'illustrazione chiara e comprensibile di come gli utenti interagiranno con il software e quali saranno i risultati di tali interazioni. La gestione e la revisione dei casi d'uso è un'attività continua. Poiché i requisiti possono evolvere nel tempo, i casi d'uso devono essere aggiornati di conseguenza per riflettere le modifiche nei requisiti e per garantire che il software finale risponda sempre alle reali necessità degli utenti.
Un caso d'uso include i seguenti elementi:

1. **Identificativo** nel formato: **UC [Numero caso d'uso] . [Numero sotto caso d'uso] - [Titolo]** (es. UC1.2-Lorem Ipsum). dove:
   - **Numero caso d'uso:** Identificativo del caso d'uso;
   - **Numero sotto caso d'uso:** Identificativo del sottocaso d'uso.
   - **Titolo**: Titolo del caso d'uso.
2. ***Attore*$_G$ principale:** Entità che interagisce attivamente con il sistema;
3. ***Attore*$_G$ secondario:** Eventuale entità esterna che non interagisce attivamente con il sistemaG, ma che permette di soddisfare il bisogno dell'attore principale.
4. **Descrizione:** Descrizione breve della funzionalità;
5. **Scenario principale:** Sequenza di eventi che si verificano quando un attore interagisce con il sistemaG per raggiungere l'obiettivo (postcondizioni) del caso d'uso;
6. **Estensioni:** Eventuali scenari alternativi che, nel caso di condizioni diverse, portano il flusso del caso d'uso a non giungere alle postcondizioni;
7. **Precondizioni:** Stato in cui si deve trovare il sistemaG affinché la funzionalità sia disponibile all'attore;
8. **Postcondizioni:** Stato in cui si trova il sistemaG dopo l'esecuzione dello scenario principale;
9. **User story associata:** Breve descrizione di una funzionalità del software, scritta dal punto di vista dell'utente, che fornisce contesto, obiettivi e valore.

2.2.2.4 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono uno strumento visivo fondamentale per rappresentare in modo grafico le interazioni tra gli attori e il sistema, illustrando i vari flussi di lavoro e i requisiti funzionali. Questi diagrammi sono utilizzati per fornire una visione chiara e intuitiva di come il sistema dovrebbe comportarsi in risposta agli input degli utenti o di altri sistemi, offrendo una visione completa delle sue funzionalità. I diagrammi dei casi d'uso sono particolarmente utili per comunicare i reqiusiti con gli stakeholder in quanto facilitano la comprensione delle funzionalità.
Riassumento quindi i casi d'uso:

- Descrivono le funzionalità del sistema dal punto di vista dell'utente.
- Non forniscono dettagli tecnici sull'implementazione del sistema.
- Mostrano le relazioni e le interazioni tra i diversi casi d'uso.
- Forniscono una panoramica generale del funzionamento del sistema.

I principali componenti di un diagramma dei casi d'uso sono i seguenti:

- **Attori:**
  Gli attori sono rappresentati come figure stilizzate all'esterno del rettangolo che rappresenta il sistema, essi indicano l'entità che interagisce con il sistema. Possono essere persone (utenti finali, amministratori, ecc.), altri sistemi o dispositivi,

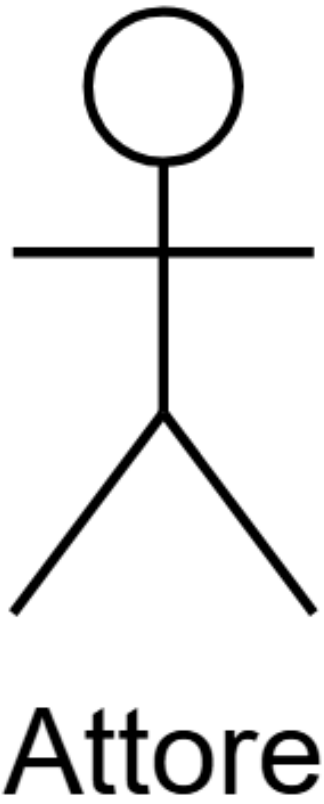

Figure 1: Esempio *Attore*$_G$

- **Casi d'Uso:**
  I casi d'uso sono rappresentati come ovali all'interno del diagramma e indicano le azioni o i comportamenti che il sistema deve eseguire in risposta agli attori. Ogni caso d'uso descrive una sequenza di azioni che il sistema segue per soddisfare un obiettivo dell'attore.

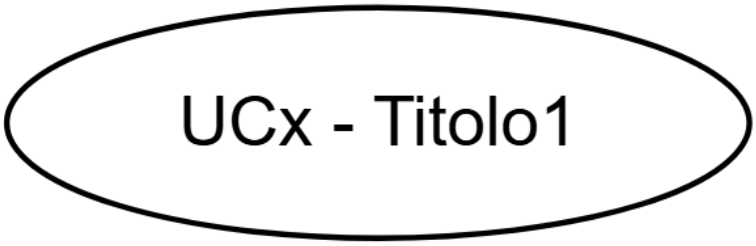

Figure 2: Esempio Caso d'uso

- **Sottocasi d'uso:** I sottocasi d'uso sono dei casi d'uso più specifici e dettaglati, questi offrono maggiori informazioni sulle funzionalità o su particolari scenari rispetto al caso d'uso principale.

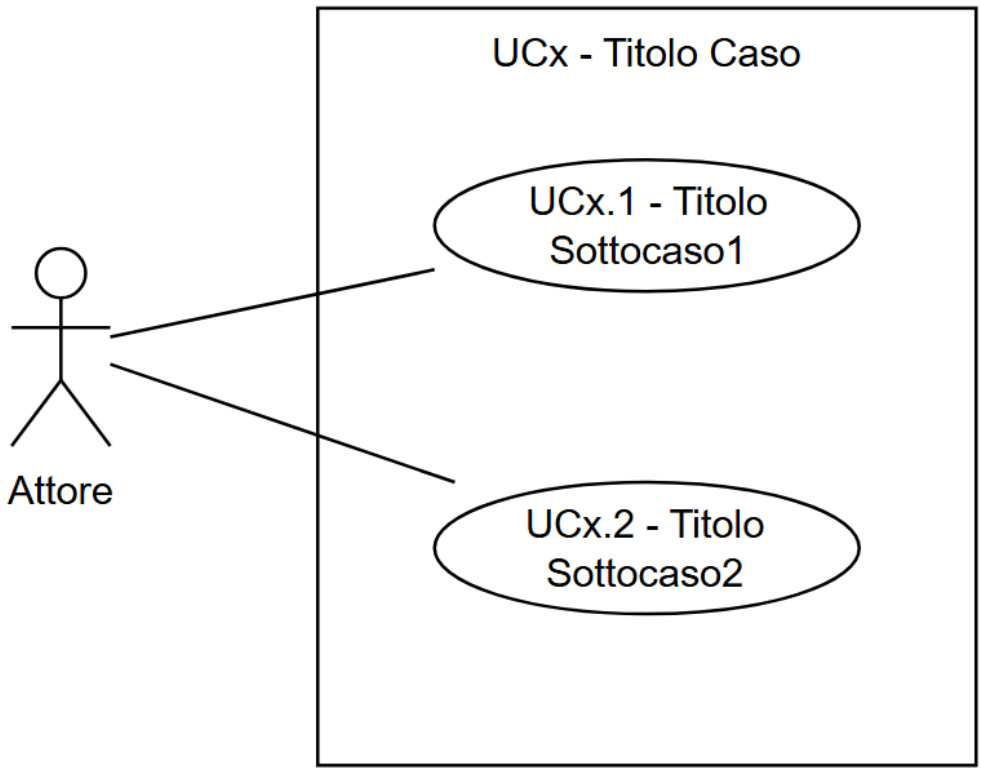

Figure 3: Esempio Sottocaso d'uso

- ***Sistema$_G$*:**
  Il sistema da modellare è tipicamente rappresentato come un rettangolo che contiene i casi d'uso. Questo confine distingue le azioni e i comportamenti che appartengono al sistema da quelli esterni ad esso. Gli attori, al di fuori di questo confine, interagiscono con il sistema.

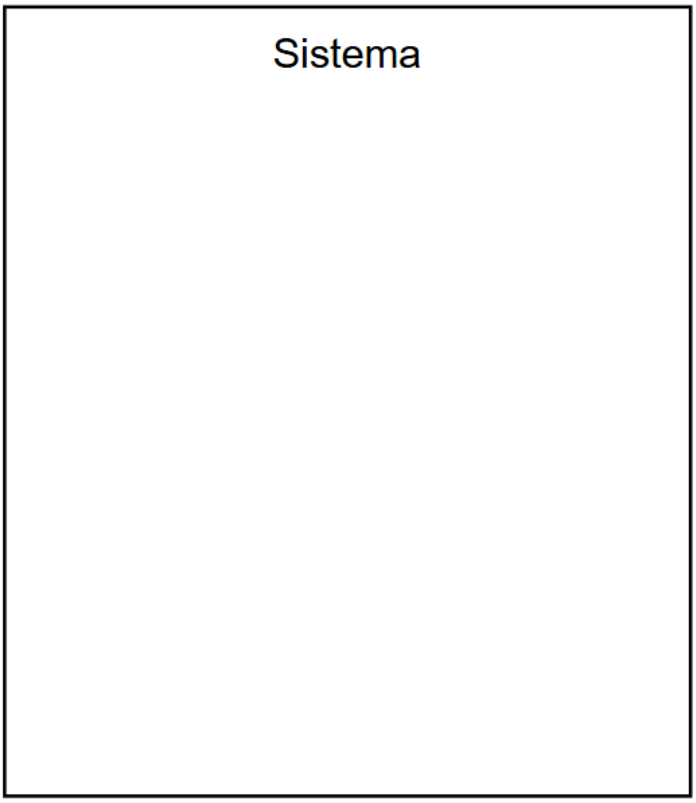

Figure 4: Esempio $Sistema_G$

- **Relazioni tra Attori e Casi d'Uso:**
  - **Associazione:**
    Una linea continua collega un attore a un caso d'uso, indicando che l'attore è coinvolto nel caso d'uso.

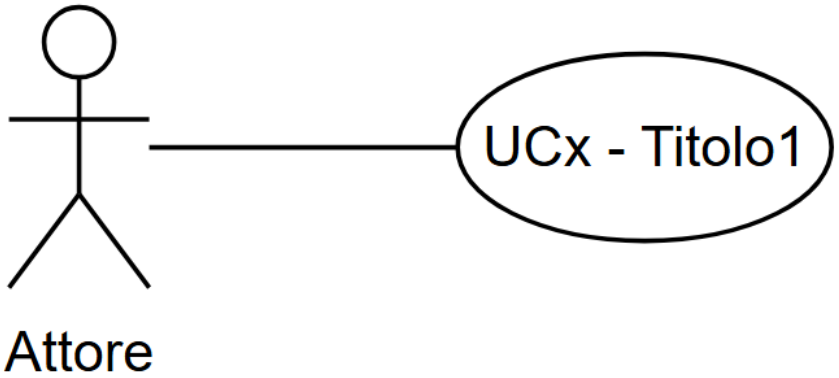

Figure 5: Esempio Associazione

- **Relazioni tra attori:**
  - **Generalizzazione tra attori:** La generalizzazione è usata per mostrare che un attore o un caso d'uso è una specializzazione di un altro, con un comportamento condiviso. Un attore o un caso d'uso derivato eredita i comportamenti e le caratteristiche di quello generale.

Figure 6: Esempio Generalizzazione tra attori

- **Relazioni tra casi d'uso:**
  - **Inclusione:** L'inclusione rappresenta una relazione tra casi d'uso dove un caso d'uso è sempre eseguito come parte di un altro caso d'uso. Questo permette di modularizzare i comportamenti comuni.

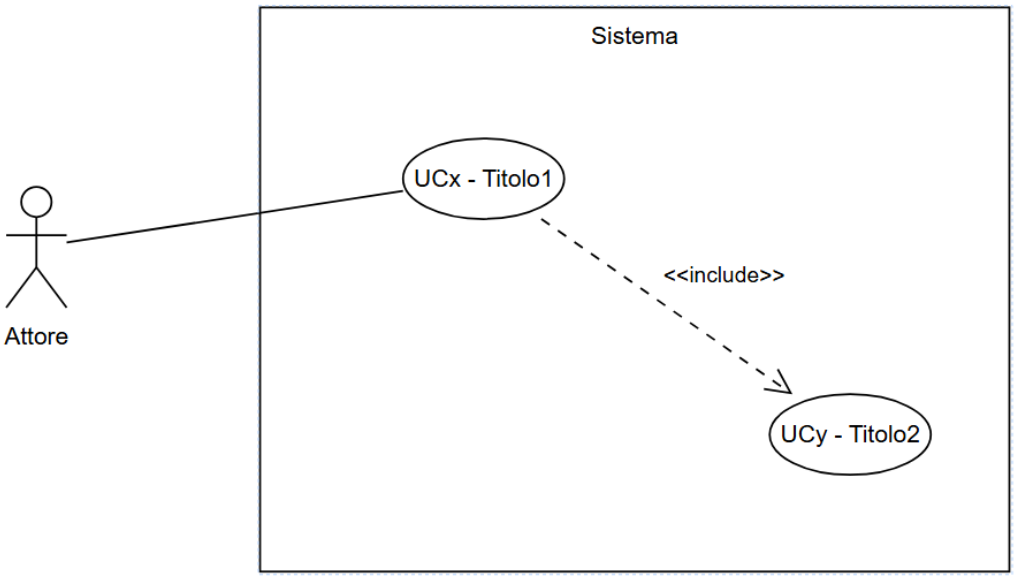

Figure 7: Esempio Inclusione

- **Estensione:** Un'istanza di estensione rappresenta un comportamento opzionale che può essere aggiunto a un caso d'uso in determinate circostanze, a seconda delle condizioni specificate.

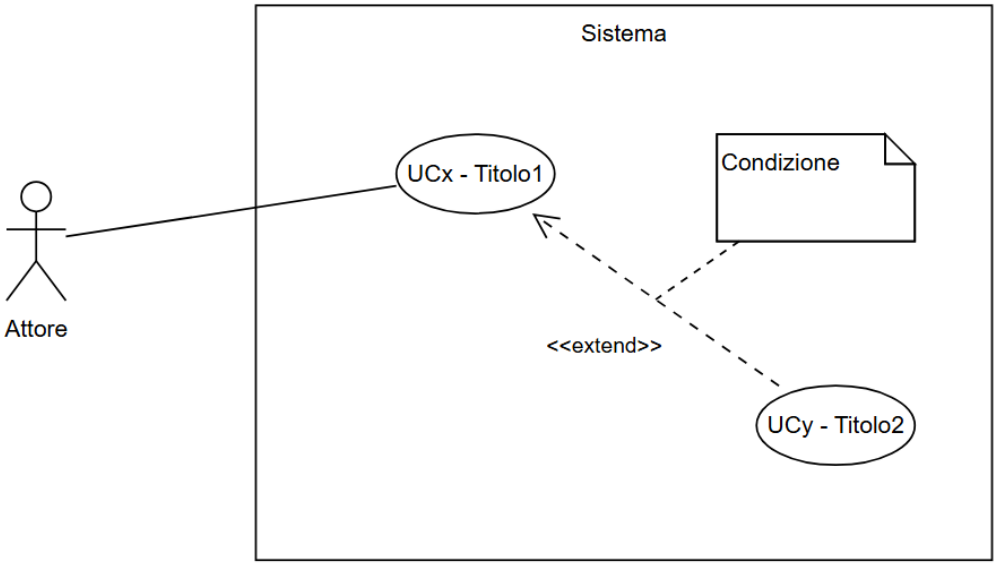

Figure 8: Esempio Estensione

- **Generalizzazione casi d'uso:** Rappresenta una relazione di ereditarietà tra casi d'uso, dove un caso d'uso più specifico eredita il comportamento da uno più generico. È simboleggiata da una linea con una freccia vuota che punta dal caso d'uso più specifico a quello generico.

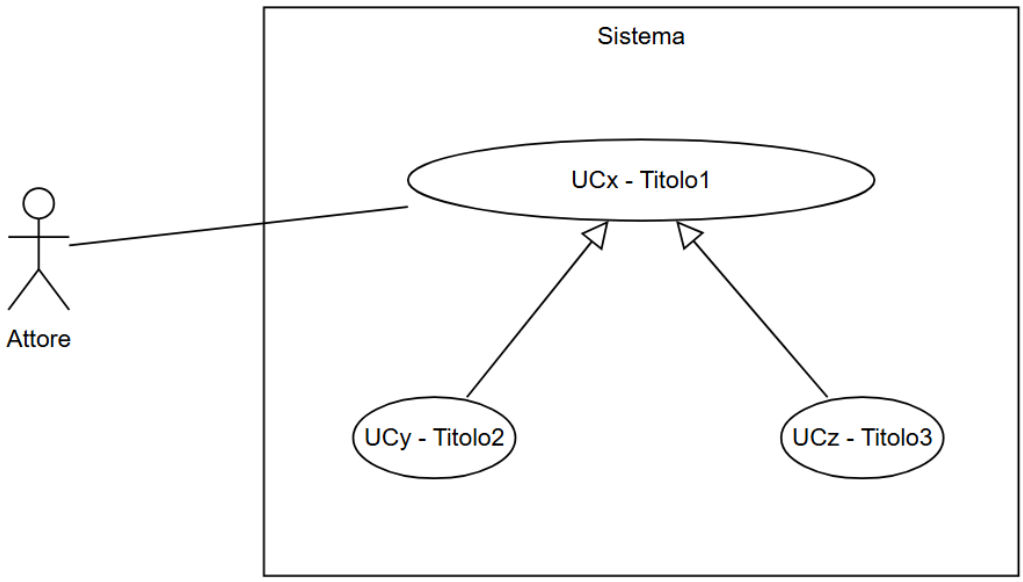

Figure 9: Esempio Generalizzazione

2.2.2.5 Requisiti

I requisiti rappresentano la base su cui si fonda l'intero processo di progettazione, sviluppo e validazione del sistema. Essi definiscono cosa il sistema deve fare, come deve comportarsi e le condizioni che devono essere soddisfatte per garantire che il prodotto finale risponda alle necessità degli utenti e degli stakeholder. I requisiti forniscono una guida chiara e misurabile per il team di sviluppo e per i tester. Garantiscono che il sistema risponda alle esigenze degli utenti e agli obiettivi del progetto. I requisiti possono essere suddivisi in diverse categorie, ognuna delle quali ha un focus specifico:

- **Requisiti funzionali:** I requisiti funzionali descrivono le funzionalità specifiche che il sistema deve supportare. Essi definiscono il comportamento del software in termini di azioni, operazioni e risposte a determinate condizioni.
- **Requisiti non funzionali:** Questi requisiti definiscono le caratteristiche qualitative del sistema, come prestazioni, sicurezza, usabilità, compatibilità, scalabilità, manutenibilità e affidabilità. Mentre i requisiti funzionali descrivono "cosa" il sistema deve fare, i requisiti non funzionali spiegano "come" il sistema deve comportarsi.

La definizione dei requisiti deve essere chiara ed esaustiva in modo da rispondere pienamente alle aspettative del committete o del proponente.
Ogni requisito è composto dai seguenti elementi:

1. **Identificativo** nel formato:

   R [Abbreviazione tipologia requisito] [Codice]

   con:

   - **Abbreviazione tipologia requisito:**
     - **RF:** Requisito funzionale;
     - **RQ:** Requisito qualitativo;
     - **RV:** Requisito di vincolo;
   - **Codice:** Numero progressivo all'interno della categoria di requisito.

2. **Importanza:**

- **Obbligatorio:** Essenziale per gli stakeholder;
- **Desiderabile:** Non indispensabile ma auspicabile in quanto aggiungerebbe valore al sistema;
- **Opzionale:** Utile e/o discutibile, da valutare in fase successiva.

3. **Descrizione:** *Dettaglio*$_G$ chiaro e completo che illustra il comportamento o la caratteristica che il software deve possedere;
4. **Fonte:** Fonte del requisito;
5. **Casi d'uso:** Elenco casi d'uso associati.

2.2.2.6 Metriche

Le metriche nell'analisi dei requisiti permettono di monitorare e misurare vari aspetti del sistema, della sua progettazione e delle sue prestazioni, aiutando i team di sviluppo a garantire che il software soddisfi i requisiti stabiliti e che rispetti gli standard di qualità predefiniti. Le metriche sono utilizzate per analizzare l'efficienza del processo di sviluppo, la qualità del codice, le prestazioni del sistema e la soddisfazione degli utenti finali. Vengono utilizzate per le seguenti motivazioni:

- **Migliorare la qualità**: ci aiutano a prevnire problemi durante lo sviluppo del sistema trovando errori, omissioni o incongruenze nei requisiti, permettendoci di correggerli.
- **Facilitare la comunicazione**: Le metriche forniscono un linguaggio comune per discutere e valutare i requisiti, semplificando la comunicazione con gli stakeholder.
- **Previsione sviluppo**: Le metriche possono essere utilizzate per prevedere quanto lavoro sarà necessario per implementare i requisiti, aiutandoci a pianificare e gestire il progetto in modo più efficiente.
- **Monitorare l'avanzamento del progetto**: ci consentono di tracciare quanti requisiti sono stati implementati, aiutandoci a individuare eventuali ritardi o difficoltà.

2.2.3 Progettazione

2.2.3.1 Descrizione

L'attività di progettazione consente di tradurre i requisiti del sistema, ottenuti nella fase precedente, in un piano d'azione concreto per una soluzione realizzativa ottimale che soddisfi le esigenze degli stakeholder. Questo piano comprende la creazione dell'architettura del sistema, la definizione delle interazioni tra i suoi componenti, e la progettazione dei dettagli di implementazione che guideranno i programmatori nella fase di codifica. La progettazione è la fase cruciale in cui vengono prese le decisioni più importanti sul comportamento e la struttura del sistema, influenzando direttamente la sua qualità, manutenibilità e performance. Questo permette di:

- perseguire la correttezza per costruzione piuttosto che per correzione
- gestire in modo efficiente la complessità del prodotto per tutta la durata dello sviluppo

2.2.3.2 Obiettivi

Gli obiettivi della fase di progettazione sono essenziali per garantire che il sistema software che viene sviluppato soddisfi i requisiti definiti nella fase di analisi e rispetti i vincoli tecnici, economici e di tempo stabiliti. La progettazione è il momento in cui vengono prese decisioni fondamentali che influenzeranno la qualità del prodotto finale, la sua manutenzione e l'evoluzione futura. Gli obiettivi della progettazione vanno oltre la semplice creazione di un sistema funzionante, cercando di garantire che il software sia qualitativamente elevato, facilmente manutenibile, scalabile e sicuro, oltre a essere realizzato nel rispetto dei vincoli di tempo e budget. La progettazione è, dunque, un momento chiave per assicurare la solidità del progetto e il successo a lungo termine del software sviluppato.
Gli obiettivi principali di questa fase possono essere sintetizzati nei seguenti punti:

- Tradurre i requisiti in soluzioni concrete: L'obiettivo principale della progettazione è quello di tradurre i requisiti funzionali e non funzionali in una struttura ben definita che possa essere realizzata e testata. Questo implica la definizione dell'architettura del sistema, la suddivisione in moduli e la pianificazione dei flussi di dati e delle interazioni tra i componenti. Ogni requisito deve trovare una soluzione nel sistema progettato, garantendo che la qualità e le aspettative degli utenti vengano soddisfatte.
- Definire l'architettura del sistema: Dopo la selezione delle tecnologie più appropriate, si passa alla progettazione di un'architettura di alto livello per definire e comprendere la struttura complessiva del prodotto, creando così una base iniziale per lo sviluppo del Proof of Concept (PoC_G), elemento cruciale della Technology Baseline, per testare le scelte effettuate in merito all'architettura e alle tecnologie selezionate, oltre a verificarne la conformità con gli obiettivi e le specifiche del progetto. La progettazione ad alto livello si concentra sull'architettura complessiva del sistema, definendo la struttura dei moduli e come essi interagiranno tra loro. Un'architettura ben progettata deve essere scalabile, flessibile e in grado di rispondere alle evoluzioni future del sistema. Deve anche tener conto dei vincoli tecnologici, dei tempi di sviluppo e delle risorse disponibili. Infine, si proseguirà con ulteriori iterazioni, effettuando miglioramenti, correzioni e integrazioni, fino a raggiungere un design definitivo. Questo design sarà fondamentale per lo sviluppo dell'MVP (Minimum Viable Product), una versione essenziale e funzionante del prodotto, che farà parte della Product Baseline.
- Ottimizzare la qualità del software: L'architettura e la progettazione dettagliata devono garantire che il sistema sia robusto, affidabile e facilmente manutenibile. La qualità del software non riguarda solo il codice, ma anche la sua capacità di rispondere a variabili come la performance, la sicurezza, la facilità di uso e l'accessibilità. Durante questa fase, si definiscono le modalità di gestione degli errori, dei guasti e delle situazioni critiche.
- Contenere i costi e rispettare i vincoli temporali: La progettazione deve essere anche efficiente dal punto di vista economico, cercando di minimizzare i costi di sviluppo senza compromettere la qualità del software. In particolare, è necessario rispettare i vincoli temporali stabiliti, pianificando

attentamente le fasi del progetto e cercando soluzioni progettuali che permettano di ridurre il tempo di sviluppo, evitando ritardi o sprechi di risorse.

2.2.4 Codifica

2.2.4.1 Descrizione e Scopo

La codifica è la fase in cui la progettazione del sistema, precedentemente definita, prende vita attraverso la scrittura effettiva del codice sorgente. In questa fase, gli sviluppatori traducono le specifiche progettuali in linguaggi di programmazione, creando il software che soddisferà i requisiti funzionali e non funzionali del sistema. Questa fase richiede precisione, attenzione ai dettagli e aderenza alle linee guida stabilite durante la fase di progettazione. È anche una fase in cui gli sviluppatori devono considerare non solo la corretta implementazione delle funzionalità, ma anche l'efficienza, la sicurezza e la manutenibilità del codice. Verranno applicate le best practices per la creazione del codice, garantendone la mantenibilità, la leggibilità e la comprensibilità secondo il "Clean Code". Tutto queso verrà fatto uniformandosi alle metriche definite nel Piano di Qualifica.

2.2.4.2 Aspettative

Ci si aspetta un software che sia in grado di soddisfare le pretese e le esigenze della proponente, il codice dovrà essere fedele alle specifiche e dovrà avere le prestazioni ottimizzate. Il codice dovrà anche essere accompagnato dai relativi test che ne dovranno garantire il corretto funzionamento e l'esattezza.

2.2.4.3 Norme di codifica

Le norme di codifica sono un insieme di linee guida e best practices che gli sviluppatori devono seguire durante la scrittura del codice. Queste norme furono enunciate da Robert C. Martin nel libro "Clean Code". Esse assicurano che il codice sia leggibile, mantenibile, e facilmente comprensibile da altri sviluppatori, anche in futuro. Le norme utilizzate dal gruppo sono le seguenti:

- **Nomi significativi:** I nomi delle variabili, delle classi e delle funzioni devono essere significativi e autoesplicativi, in modo da rendere chiaro il loro scopo e il loro utilizzo.
- **Indentazione:** Utilizzare uno stile di indentazione uniforme per facilitare la lettura e la comprensione del codice.
- **Lunghezza delle funzioni:** Le funzioni devono essere brevi e focalizzate su un'unica responsabilità, in modo da rendere il codice più leggibile e manutenibile.
- **Commenti:** Utilizzare commenti per spiegare il codice complesso o per fornire informazioni aggiuntive, evitando commenti ridondanti o superflui.

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

I container offrono un ambiente isolato e leggero che consente di eseguire applicazioni in modo coerente, indipendentemente dall'ambiente sottostante. Le regole utilizzate dai programmatori per la scrittura dei file Docker sono le seguenti:

- **Scrivere file ottimizzati:** minimizzare il numero di istruzioni per minimizzare i layer dell'immagine;
- **Utilizzare nomi descrittivi per i container: i nomi dei container** devono riflettere in modo chiaro il loro contenuto o la loro funzione;
- **Utilizzare versioni specifiche:** specificare sempre la versione dell'immagine di base utilizzata nel Dockerfile, evitando l'uso di "latest";
- **Utilizzare immagini ufficiali:** utilizzare immagini ufficiali o verificate da fonti affidabili;
- **Evitare processi** con privilegi elevati;
- **Utilizzo variabili ARG**: Utilizzare variabili ARG per gestire informazioni sensibili;
- **Ottimizzare le risorse**: Impostare dei limiti di utilizzo di risorse.

- **Utilizzare variabili d'ambiente**: Utilizzare variabili d'ambiente per configurazioni dinamiche, fornendo valori predefiniti;
- **Configurazione loggin**: Configurare i container per registrare efficacemente i log, integrando strumenti di monitoraggio se necessario;
- **Integrazione *Test*$_G$**: Integrare i test nell'immagine del container;

2.2.5.2 Strumenti

Gli strumenti principali utilizzati nel processo includono:

- **Microsoft teams**: Un servizio di videoconferenza per gli incontri e una piattaforma di messaggistica istantanea per le comunicazioni con la proponente;
- ***Git*$_G$ Flow**: Strategia di branching adottata dal team che facilita la gestione delle release e lo sviluppo collaborativo tramite l'utilizzo workflow strutturati
- **GitHub**: Piattaforme online che offrono repository *Git*$_G$ collaborative.
- ***Visual Studio Code*$_G$**: Editor di codice che supporta diversi linguaggi di programmazione.
- **GitHub *Copilot*$_G$**: Strumento che usa l'IA che assiste nella scrittura del codice.
- **Docker**: *Piattaforma*$_G$ per la containerizzazione attraverso le immagini docker, usata per garantire la portabilità e la consistenza degli ambienti di sviluppo e produzione.
- ***Supabase*$_G$**: Alternativa open-source a Firebase per gestione di autenticazione e storage (con database), facilitando la gestione del backend.
- **Ollama + DeepSeek**: Vengono usati per l'esecuzione locale di modelli *AI*$_G$ e DeepSeek per elaborazione avanzata del linguaggio naturale, usato per rispondere alle domande degli utenti.

3 *Processi*$_G$ di supporto

3.1 Documentazione

3.1.1 Introduzione

La documentazione è l'informazione che accompagna un prodotto software. Descrivere il prodotto a sviluppatori, distributori e utenti, in modo da facilitare l'attività di sviluppo e da permettere una comprensione del prodotto stesso senza un supporto umano.
Grazie alla documentazione, vengono dettate regole precise e dirette che assicurano che i processi si svolgano con la qualità attesa, e la semplificazione della manutenzione del prodotto.

Questa sezione definisce delle procedure ripetibili che hanno l'obbiettivo di uniformare la documentazione e di semplificare la redazione dei documenti. Tali procedure dovranno essere applicate da tutti i membri del team. I file sorgenti LaTeX con tale documentazione saranno inseriti nella repository privata DocsSource.

3.1.2 Ciclo di vita

Ogni documento segue le seguenti fasi:

1. Si crea un branch per lo sviluppo del documento/sezione nella repository DocsSource attraverso i seguenti comandi:

```
git checkout develop
git flow feature start id_FEATURE
```

2. Si copia all'interno della cartella appropriata il template del documento/sezione da redigere;
3. Si redige il/la documento/sezione assegnata.
4. Si aggiorna il registro delle modifiche, incrementando la versione e aggiungendo i dati in una nuova riga. Quest'azione deve seguire la convenzione descritta nella sezione X.Y.Z e tramite "X.Y.Z".
5. Si rende accessibile il branch nella repository attraverso i comandi:

```
git add .
git commit -m "Descrizione commit"
git push origin feature/Id_FEATURE
```

6. Si segnala il completamento dell'attività spostando l'issue nella colonna "In review"
7. Si crea una Pull Request, selezionando come branch di destinazione quella "main" e come branch di origine quello dedicato alla redazione del documento, o della sezione.;
8. Si clicca su "Create Pull Request"
9. Si inserisce un titolo significativo e una breve descrizione, si selezionano i verificatori su "Reviewers" e infine si clicca su "Create Pull Request".
10. Per modifiche richieste dal verificatore, o dai verificatori, si ripetono i punti, in ordine, dal punto 3 al punto 5;
11. Si elimina il branch quando la pull request viene chiusa o risolta.

3.1.3 Sorgente documenti

Per consentire a più Redattori di lavorare su un singolo file, è stato decidere di creare singoli file per ogni sezione/sottosezione individuata dal Responsabile. Così facendo, si evita la modifica di parti del documento che non sono assegnate al singolo *Redattore*$_G$.
Tutti questi file vengono collegati in un file principale col nome del documento. Il collegamento verrà effettuato mediante il comando `\input{Sezione.tex}` o `\input{Sottosezione.tex}`.
Nel caso dei verbali non si usa questo approccio date le sezioni brevi.

3.1.4 Nomenclatura dei documenti

Il nome dei documenti deve essere omogeneo: la prima lettera deve essere minuscola, mentre le restanti minuscole. Tranne per i verbali, nel nome del documento deve essere riportata anche la versione attuale del documento (vedi convenzione per il versionamento, sezione X.Y.Z).
Nello specifico la convenzione da seguire è la seguente:

- **Verbali:**
  - **Interni:** VerbaleInterno_AAAAMMDD;
  - **Esterni:** VerbaleEsterno_AAAAMMDD;
- **Norme di *Progetto*$_G$**: Norme_di_progetto_vX.Y.Z;
- **Analisi dei requisiti**: Analisi_dei_requisiti_vX.Y.Z;
- **Piano di progetto**: Piano_di_progetto_vX.Y.Z;
- ***Glossario*$_G$**: *Glossario*$_G$_vX.Y.Z.

3.1.5 Struttura del documento

I documenti ufficiali seguono una struttura precisa e comune, rigorosamente rispettata.

3.1.5.1 Prima pagina

Nella prima pagina è presente:

- Logo, nome e mail del gruppo;
- Nome del documento;
- Redattori;
- Verificatori;
- Amministratore;
- *Destinatari*$_G$;
- Firma digitale del responsabile.

3.1.5.2 Registro delle modifiche

Ogni documento deve essere provvisto di un registro delle modifiche tabellare che contiene un riassunto delle modifiche apportate al documento nel corso del tempo.
La tabella viene inserita nella sezione registro delle modifiche, prima dell'indice. All'interno vengono riportate le seguenti informazioni:

- *Versione*$_G$ del file;
- *Data*$_G$ di rilascio;
- *Autore*$_G$;
- *Verificatore*$_G$;
- Descrizione: sintesi delle modifiche apportate.

Informazioni utili:

- La prima riga della tabella identifica la versione attuale del documento. Per permetterne l'integrazione automatica nel nome del Documento, durante la compilazione del file . tex, è necessario utilizzare il comando:
  `{\label{Git_Action_Version} X.Y.Z.}`.
- La convenzione per il versionamento è presente alla sezione 3.*.*

3.1.5.3 Indice

Ogni documento deve contenere un indice, posto nella pagina seguente al Registro delle modifiche. All'interno saranno elencate le sezioni e le sottoszioni, ognuna delle quali raggiungibile tramite click.

3.1.5.4 Piè di pagina

In ogni piè di pagina deve essere presente il logo del gruppo e il numero di pagina.

3.1.5.5 Verbali

Questi documenti vogliono riportare gli oggetti di discussione, le decisioni adottate, le azioni da intraprendere e le figure coinvolte, durante i meeting.
Si distinguono in verbali interni ed esterni. I primi sono destinati all'uso interno dell'organizzazione, mentre i secondi trovano applicaione quando nelle discussioni o nelle decisioni vengono coinvolte terze parti.
La struttura dei verbali è la seguente:

- **Prima pagina** Oltre alle informazioni comuni a ogni documento vengono riportate:
  - *Data*$_G$ e tipologia della riunione nel nome del documento
  - Luogo fisico o Canale di comunicazione adottato

  - Ora di inizio e fine
  - Partecipanti della riunione (interni ed in caso esterni)
  - Il Responsabile

- **Corpo del documento** Sono presenti le sezioni:
  - **Revisione del periodo precedente** Viene analizzato lo stato delle attività, aspetti positivi ed eventuali difficoltà incontrate, in modo da identificare azioni migliorative per ottimizzare i processi.
  - **Ordine del giorno** Vengono elencate le tematiche discusse durante la riunione, con le relative decisioni.
    Se sono presenti richieste di chiarimenti efftuate verso le terze parti coinvolte, vengono incluse nella sezione "**Richieste e chiarimenti**".
  - ***Attività*$_G$ da svolgere** Tabella in cui vengono identificate le attività e in cui si specifica:
    * Nome della task
    * Id issue
    * *Verificatore*$_G$ a cui è destinata l'attività

3.1.6 Regole stilistiche

Regole sintattiche:

- I titoli delle sezioni iniziano con la lettera maiuscola;
- Le date vengono scritte nel formato GG/MM/AAAA (giorno/mese/anno);
- Ogni voce dell'elenco deve terminare con un punto e virgola (**;**), tranne l'ultima che, invece, deve terminare con un punto (**.**);
- I numeri razionali devono essere scritti con la virgola;
- Quando si fa riferimento a informazioni contenuti in documenti diversi, si devono indicare questi ultimi, con la relativa versione e sezione. (Ex: Norme di *Progetto*$_G$ v1.0.0 sez. 1.2.3).

Stile del testo:

- **Grassetto**:
  - Titoli delle sezioni/sottosezioni/paragrafi;
  - Parole/frasi considerate di rilievo;
  - Nomi di Aziende.
- **Italico**:
  - Riferimenti a paragrafi, sezioni e documenti;
  - Termini presenti nel *Glossario*$_G$;
  - Nomi delle aziende.

Riferimenti:

- Per quelli **interni** si usa il colore RGB (R,G,B);
- Per quelli **esterni** si usa il colore: RGB (R,G,B);

3.1.7 Strumenti

Gli strumenti usati per la redazione dei documenti sono i seguenti:

- **GitHub:** serve per il versionamento, la collaborazione, la distribuzione e l'automatizzazione della compilazione;
- **LaTeX:** serve per la redazione dei documenti;
- ***Visual Studio Code*$_G$:** IDE utilizzato per la redazione dei documenti. Viene utilizzato il plugin **LaTeX Workshop**

3.2 Verifica

3.2.1 Scopo

La verifica è un processo che guida l'intero ciclo di vita software, e che garantisce efficienza e correttezza nelle attività. Il suo scopo è quello di assicurare che i prodotti siano conformi rispetto ai requisiti. Durante

la fase di Verifica viene analizzata la qualità del prodotto e dei processi. Affinché il prodotto passi alla fase successiva (quella di Validazione) è necessario seguire convenzioni chiare e dettagliate, in modo tale da garantire le qualità attese dettate da standard definiti nel documento "Piano di Qualifica".

Per permettere ai Verificatori di valutare la qualità del prodotto e attuare ogni attività di verifica, ognuna di queste viene documentata all'interno del documento Piano di qualifica.

3.2.2 Verifica dei documenti

Il *Verificatore*$_G$ quando visualizza un'*Issue*$_G$ nella colonna "In Review" della DashBoard (da mettere link), deve convalidare il file corrispondente nella repository privata DocsSource, garantendo la qualità e l'accuratezza del contenuto.

Dopo che il redattore avrà creato la Pull Request, il revisore troverà la richiesta di unire il branch dedicato alla redazione del documento o di una sua sezione, al branch "develop".
Durante questa fase il revisore dovrà esaminare ciò che è stato fatto, e in caso di necessità, potrà fornire feedback e suggerimenti ai Redattori per eventuali modifiche.

Affinché un documento passi alla fase di valutazione, devono essere fatte le seguenti verifiche:

- **Revisione tecnica**: le informazioni all'interno del documento devono essere corrette, coerenti e devono rispettare le norme stabilite;
- **Conformità alle norme**: il documento deve seguire gli standard prestabiliti per lo stile, la formattazione e la struttura;
- **Consistenza e coerenza**: il documento non deve presentare discrepanze o contraddizioni rispetto a quanto scritto all'interno, e a quanto scritto in altri documenti;
- **Revisione grammaticale e ortografica**: il documento non deve presentare errori grammaticali, ortografici e di punteggiatura;
- **Chiarezza e comprensibilità**: il contenuto e il linguaggio usato devono essere chiari e comprensibili ai destinatari.

Procedimento di convalidazione

1. **Accettazione della Pull Request**: si accede alla pagina della Pull Request, si risolvono eventuali conflitti, e infine si effettua il Merge Pull Request;
2. **Eliminanazione del branch**: si elimina il branch di feature creato per la redazione del documento o della sezione;

3. **Spostamento della issue in Done**: nella *Dashboard*$_G$(link) si sposta la relativa issue dalla colonna "In Review" a quella "Done";

Solamente quando il documento avrà raggiunto un certo livello di maturità, verrà aperta una nuova Pull Request destinata al responsabile per la sua convalida e per il suo merge all'interno del branch "main". Quest'ultimo passaggio attiverà la GitHub Action che andrà a generare automaticamente il PDF del documento, con la relativa versione, all'interno della repository "Documents". Nel caso in cui la compilazione dovesse fallire, il responsabile dovrà accedere alla sezione "Actions" di GitHub, esaminare l'origine dell'errore e correggerlo.

3.2.3 Analisi

3.2.3.1 Analisi statica

Tipologia di analisi che mira a individuare problemi o irregolarità all'interno del prodotto senza la sua esecuzione. Gli errori che intende individuare sono di tipo formale, come la presenza di difetti/proprietà indesiderati/e, o come la violazione di vincoli imposti.
Le tecniche impiegate dal gruppo sono:

Walkthrough
Metodologia che analizza sistematicamente la documentazione e/o il codice, e che promuove la collaborazione tra *Autore*$_G$ e *Verificatore*$_G$. Questo approccio è preferito durante le fasi iniziali del progetto, dove si trovano prodotti poco complessi.

Inspection
Tecnica strutturata composta da una sequenza di passaggi ben definiti, che mira a rilevare difetti specifici all'interno di un documento o del codice.
Gli elementi da verificare sono organizzati in liste di controllo, ognuna documentata all'interno del documento Piano di qualifica.

In fasi avanzate del progetto, con la presenza di prodotti complessi, la verifica di quest'ultimi risulta più efficace tramite inspection rispetto a walkthrough.

3.2.3.2 Analisi dinamica

Tipologia di analisi applicabile al solo software che verifica il corretto funzionamento del codice, attraverso la sua esecuzione. Durante tale processo vengono eseguiti una serie di test, derivati dai requisiti (funzionali e non), che assicurano l'esecuzione accurata del software e che che garantiscono risultati ottenuti corrispondenti a quelli attesi.

Per garantire efficacia, questo processo deve essere automatizzato e ripetibile, in modo da permettere una valutazione oggettiva del prodotto. La definizione e l'esecuzione dei test seguono i principi del *Modello a V*$_G$.

3.2.3.3 *Test*$_G$ di unità

Test$_G$ progettati per la verifica delle singole componenti/unità, del sistema che ne garantiscono il corretto funzionamento.
Si suddividono in 2categorie:

- ***Test*$_G$ funzionali**: verificano che ogni unità esegua le funzioni specificate nell'implementazione e che produca i risultati desiderati.
- ***Test*$_G$ Strutturali**: verificano la struttura interna, la logica di controllo e il flusso dei dati dell'unità attraverso l'esaminazione del codice.

3.2.3.4 *Test$_G$* di integrazione

Test$_G$, pianificati durante la fase di progettazione architetturale, cruciali nell'analisi dinamica del software che valutano il corretto funzionamento delle unità quando combinate. Più precisamente verificano se quest'ultime collaborano in maniera efficacie.
L'obbiettivo di tali test è di assicurare che il software funzioni perfettamente e secondo le specifiche del progetto.

3.2.3.5 *Test$_G$* di *Sistema$_G$*

Test$_G$, definiti durante l'analisi dei requisiti, che verificano la copertura dei requisiti, valutando interamente il software e accertando le corrette integrazioni tra le unità.

Vengono progettati durante l'analisi dei requisiti, ed eseguiti solo dopo il completamento dei test di integrazione.

3.2.3.6 *Test$_G$* di accettazione

Test$_G$ fondamentali per il rilascio del software. Garantiscono che quest'ultimo soddisfi le aspettative, e che risponda ai requisiti specificati.

3.2.3.7 Sequenza delle fasi dei test

La sequenza delle fasi di test è così composta:

1. *Test$_G$* di Unità;
2. *Test$_G$* di Integrazione;
3. *Test$_G$* di *Sistema$_G$*;
4. *Test$_G$* di Accettazione.

3.2.4 Classificazione dei test

Ogni test è indentificato in base alla propria tipologia e atttraverso un codice identificativo nel seguente formato:

T [Tipologia *Test$_G$*] [Codice]

dove:

- **[Tipologia *Test$_G$*]:**
  - U: test di Unità;
  - I: test di Integrazione;
  - S: test di *Sistema$_G$*;
  - A: test di Accettazione.
- **[Codice]**: numero progressivo, nel caso il test non avesse un padre; mentre nel caso in cui il test abbia un padre, avrà il seguente formato:

  [Codice.padre].[Codice.figlio]

  dove il **[Codice.padre]** identifica univocamente il padre del test; mentre **[Codice.figlio]** è il numero progressivo che identifica il test.

3.2.5 Stato dei test

Nella sezione relativa ai test nel documento Piano di qualifica per ogni test viene registrato il suo risultato.
Lo stato di un test può essere:

- **N-I**: il test non è stato implementato;
- **S**: il test ha dato esito positivo;
- **N-S**: test ha dato esito negativo.

3.2.6 Strumenti

- **Modulo "unittest"** ***Python***$_G$: liberia standard che permette di creare, organizzare ed eseguire test di unità efficaciemente e in modo automatizzato.

3.3 Validazione

3.3.1 Introduzione

Nel contesto dell'ingegneria del software, la validazione rappresenta una fase cruciale per verificare che il prodotto sviluppato soddisfi pienamente le aspettative di chi lo ha commissionato. Questo passaggio è fondamentale per accertarsi che il software risponda agli obiettivi e ai requisiti definiti nelle fasi iniziali del progetto.
Un aspetto essenziale della validazione è il confronto diretto con il committente e gli stakeholder, utile per raccogliere feedback e assicurare che quanto realizzato sia in linea con le aspettative degli utenti finali.
Il prodotto conclusivo deve quindi:

- Soddisfare tutti i requisiti concordati con il committente;
- Funzionare correttamente nell'ambiente operativo previsto.

Lo scopo finale è garantire che il prodotto sia pronto per il rilascio, segnando così il completamento del ciclo di vita del progetto.

3.3.2 Procedura di validazione

La procedura di validazione si basa sui test pianificati durante l'attività di verifica, come descritti nelle Norme di *Progetto*$_G$. L'elemento centrale di questa fase è rappresentato dal test di accettazione, che serve a confermare la conformità del prodotto rispetto ai requisiti.
I test devono garantire:

- La copertura dei casi d'uso previsti;
- Il soddisfacimento dei requisiti obbligatori;
- La conformità con gli ulteriori requisiti concordati con il committente.

3.3.3 Strumenti utilizzati

Per la validazione verranno impiegati gli strumenti definiti per le attività di verifica, scelti in base alle esigenze del progetto e alle specifiche del software.

3.4 Gestione della configurazione

3.4.1 Introduzione

La gestione della configurazione è un processo che norma il tracciamento e il controllo delle modifiche a documenti e prodotti del software. È fondamentale nell'ambito di sviluppo software eviene applicata a

qualsiasi "artefatto". Assicura funzionamento del sistema nonostante modifiche che potrebbero essere apportate.

Secondo lo standard ISO/IEC12207:1995 è un processo di identificazione, organizzazione e controllo delle modifiche apportate agli artefatti. Secondo lo standard IEEE 828-2018, invece, viene definito come "un processo disciplinato per gestire l'evoluzione del software".

3.4.2 Versionamento

Per identificare la versione di un documento si usa il formato X.Y.Z, dove:

- **X:** incrementato al raggiungimento di RTB_{GG}, PB_{GG} ed eventualmente CA_{GG};
- **Y:** incrementato quando vengono apportate modifiche significative al documento. Questi possono essere cambiamenti strutturali, nuove sezioni o modifiche sostanziali;
- **Z:** incrementato per modifiche minori al documento o aggiornamenti, come correzioni di errori, miglioramenti minimi o l'aggiunta di contenuti non particolarmente rilevanti.

Ogni variazione deve essere presente nel registro delle modifiche. L'incremento dei valori più significativi porta quelli minori a zero. Ad esempio nel caso di un documento alla versione 0.7.5, qualora venisse raggiunta la RTB_G la versione verrebbe incrementata a 1.0.0.

3.4.3 *Repository*$_G$

Il gruppo utilizza **GitHub**, una piattaforma basata sullo strumento di controllo versione distribuita Git_G.
Le repository del gruppo **Byte Your Dreams** sono le seguenti:

- DocsSource: repository privata per la gestione, lo sviluppo e il versionamento dei file sorgente della documentazione;
- Documents: repository destinata ai committenti/proponente, che contiene i file in formato PDF riguardanti la documentazione, ottenuti dalla compilazione automatizzata dei file in formato TeX della repository DocsSource;
- Proof-of-Concept: repository destinata al PoC_G.

All'interno della repository viene utilizzato **Gitflow** come stile di flusso di lavoro.

3.4.3.1 Flusso generale Gitflow

- $Branch_G$ `main`: branch principale della repository;
- $Branch_G$ `develop`: branch utilizzato per lo sviluppo, creato a partire dal branch `main`;
- $Branch_G$ `feature`: branch utilizzato per lo sviluppo di nuove funzionalità/miglioramenti, creato a partire dal branch `develop`;
- $Branch_G$ `release`: branch che prepara il software per il rilascio, creato da branch `develop`;
- $Branch_G$ `hotfix`: branch creato dal `main` se si verificano dei problemi;
- Merge di `develop` in `main`: fusione del branch `develop` in quello `main`. Viene effettuato quando le funzionalità completate sono state verificate e validate.
- Merge di `feature` in `develop`: fusione del branch `feature` in quello `develop`. Viene effettuato quando la funzionalità viene completata.
- Merge di `release` in `develop` e `main`: a seguito del completamento del branch `release`, questo viene unito sul branch `develop` e `main`, segnando un rilascio stabile;
- Merge di `hofix` in `develop` e `main`: a seguito della risoluzione del problema, il branch `hotfix` viene fuso con i branch `develop` e `main`.

Comandi

- Inizializzazione Gitflow:

```
git flow init
```

- Sviluppo di una feature:

```
git flow feature start nomeFeature
```

```
git flow feature finish nomeFeature
```

- Rilascio di una versione:

```
git flow release start X.Y.Z
```

```
git flow release finish X.Y.Z
```

- Risoluzione di un bug:

```
git flow hotfix start nomeHotfix
```

```
git flow hotfix finish nomeHotfix
```

- Pubblicazione modifiche:

```
git push origin develop
git push origin master
git push origin feature/nomeFeature
git push origin release/X.Y.Z
git push origin release/nomeHotfix
```

3.4.4 Struttura repository "Documents"

La repository "Documents" è organizzata nel seguente modo:

- **Documenti Esterni:**
  - **Verbali:** contiene tutti i verbali redatti dall'inizio del progetto, che vedono la presenza di figure esterne;
  - **Analisi dei requisiti v1.0.0**;
  - **Valutazione dei costi e assunzione impegni**.
  - **Piano di qualifica v1.0.0**;
  - **Piano di progetto v1.0.0**;
- **Documenti interni:**
  - **Verbali:** contiene tutti i verbali redatti dall'inizio del progetto, che vedono la presenza dei soli membri del gruppo;
  - **Valutazione dei capitolati**;
  - ***Glossario*$_G$ v1.0.0**;
  - **Norme di progetto v1.0.0**;
- Lettera di presentazione
  - di Candidatura
  - *RTB*$_G$

3.4.5 Sincronizzazione e Branching

3.4.5.1 Documentazione

Per il processo di documentazione, viene creato un branch per ogni documento/sezione da redigere. Questo nel caso in cui due membri lavorano allo stesso documento, permette di non andare incontro a conflitti.

Convenzione per la nomenclatura dei branch relativi alla redazione/modifica di documenti

- Deve essere presente l'identificativo del documento da redarre/modificare. Di seguito è riportata una tabella che descrivere gli id:

Documento	Id
VerbaleEsterno	VE
VerbaleInterno	VI
Norme di progetto	NP
Analisi dei requisiti	AR
Piano di progetto	PP
Piano di qualifica	PQ

- Per i verbali, si mette un underscore e la data dopo l'ID: `IdDocumento_AAAAMMGG`
- Per le sezione dei documenti il norme deve essere così composto: `IdDocumento_NomeSezione`

3.5 Risoluzione dei problemi

3.5.1 Introduzione

Per analizzare e sanare le problematiche riscontrate durante lo sviluppo del software, viene messo in atto il processo di risoluzione dei problemi, il quale ha come obiettivo quello di garantire che tali problemi individuati vengano gestiti e risolti il prima possibile.
Durante questo processo vengono analizzate le problematiche emerse durante il ciclo di vita del software, identificandone le cause, in modo da poter prevenire rischi futuri. L'obiettivo è anche quello di garantire un miglioramento costante del prodotto che viene sviluppato, ottimizzandone i processi.
Il processo di risoluzione dei problemi prevede attività strutturate come l'analisi delle cause, la valutazione degli impatti dei problemi riscontrati, e la definizione di azioni correttive future.
E' necessario che vengano documentati tutti i problemi riscontrati e le relative soluzioni in modo da garantire la tracciabilità degli stessi per avere un controllo nel tempo.

3.5.2 Gestione dei rischi

I rischi legati al progetto sono stati identificati e inseriti nella sezione "Analisi dei rischi" del documento Piano di *Progetto*$_G$. In questa sezione ad ogni rischio viene associata la sua probabilità di occorrenza e la misure da prendere per la loro prevenzione.
Vengono definite le strategie da mettere in atto per mitigare l'impatto che tali rischi possono avere nell'esecuzione dei vari processi del progetto.

3.5.2.1 Metriche

Metrica	Nome
M11RNP	Rischi non previsti

Table 1: Metriche relative alla gestione dei processi

3.6 Gestione della qualità

3.6.1 Introduzione

L'obiettivo delle attività eseguite durante il processo di gestione della qualità è quello di garantire la qualità dei prodotti sviluppati dai fornitori e di essere in grado di soddisfare e rispettare i requisiti specificati dal proponente. Tra le attività da svolgere bisogna identificare le metriche e i criteri di qualità, pianificare le azioni da mettere in atto per garantire e controllare la qualità, e verificare costantemente attraverso revisioni e test il prodotto che si sta sviluppando. La gestione della qualità mira a garantire che il prodotto software è conforme alle aspettative degli utenti e ai requisiti del progetto.

La gestione della qualità è quindi un processo che segue l'intero ciclo di vita del software con l'obiettivo che il prodotto finale rispetti gli standard di qualità predefiniti.

3.6.2 *Attività$_G$*

Le attività che il team si impegna a svolgere per garantire la qualità dei processi e dei prodotti software sviluppati sono le seguenti:

1. **Definizione degli *Standard$_G$* di qualità:**

   il processo di gestione della qualità inizia con la definizione degli standard di qualità che il software deve raggiungere. Questi standard possono includere requisiti funzionali e non funzionali;

2. **Pianificazione della qualità:**

   In un piano di qualità vengono identificate attività e risorse che garantiscono la qualità del prodotto durante il ciclo di vita del progetto;

3. **Assicurazione della qualità:**

   Durante il ciclo di vita del progetto vengono svolte attività di monitoraggio per garantire che i processi siano conformi agli standard di qualità predefiniti;

4. **Controllo della qualità:**

   il controllo della qualità prevede che vengano effettuati dei test che mirano a garantire che il prodotto software sviluppato è conforme agli standard di qualità, e risponde ai requisiti richiesti;

5. **Gestione delle modifiche:**

   viene fatto un controllo sulle modifiche attraverso un registro delle modifiche che ha come obiettivo quello di controllare l'evoluzione di un prodotto. Questo garantisce che la qualità venga gestita anche attraverso le modifiche che vengono effettuate;

6. **Miglioramento continuo e correzione:**

   attraverso la gestione della qualità viene fatto un monitoraggio continuo durante il ciclo di vita del prodotto software per permettere un miglioramento costante dei processi;

7. **Coinvolgimento degli stakeholder:**

   durante il processo di gestione della qualità vengono coinvolti anche gli stakeholder che, inviando feedback, aiutano a controllare che il prodotto rispetti le aspettative;

8. **Formazione e competenza del team:**

   per garantire un prodotto software di qualità, il team deve costantemente studiare le tecnologie e le norme da mettere in pratica per poter rispondere alle esigenze degli stakeholder.

3.6.3 Piano di qualifica

Per effettuare il controllo della qualità, le attività di pianificazione della stessa vengono descritte nel documento Piano di Qualifica, il quale è di notevole importanza nel processo di gestione della qualità. In questo documento vengono definite in dettaglio le specifiche di qualità relative al prodotto software.

3.6.4 PDCA

Nell'ambito dell'attività di miglioramento continuo e correzione, si è scelto di adottare il ciclo PDCA, conosciuto anche come ciclo di Deming.
Il ciclo PDCA è una metodologia iterativa che consente il controllo e il miglioramento continuo dei processi e dei prodotti. Affinché si possa ottenere un miglioramento effettivo, bisogna attuare le seguenti 4 fasi:

- **Plan**
  Consiste nello svolgere le attività di pianificazione necessarie per stabilire quali processi debbano essere avviati e in quale sequenza, al fine di raggiungere obiettivi specifici;
- **Do**
  Consiste nell'effettiva esecuzione di quanto pianificato, raccogliendo dati e misurando i risultati durante lo svolgimento delle attività;
- **Check**
  Consiste nell'analisi e nell'interpretazione dei dati raccolti durante l'esecuzione (Do). Questi dati vengono valutati utilizzando metriche prestabilite e confrontati con gli obiettivi previsti nella fase di Plan.
- **Act**
  Si consolida quanto di positivo è stato rilevato nella fase precedente (Check) e si implementano le strategie correttive necessarie per migliorare gli aspetti che non hanno raggiunto i risultati attesi, analizzandone approfonditamente le cause. In questo modo, il ciclo PDCA viene continuamente perfezionato e ogni iterazione successiva aggiunge valore al processo.

3.6.5 Strumenti

Gli strumenti utilizzati nel processo di gestione della qualità sono rappresentati dalle metriche.

3.6.6 Struttura e identificazione metriche

- **Metrica:**
  identificativo della metrica nel formato:

  M [numero] [abbreviazione]

  dove:

  - `M`: sta per metrica;
  - `[numero]`: numero progressivo univoco per ogni metrica;
  - `[abbreviazione]`: abbreviazione composta dalle iniziali del nome della metrica.
- **Nome:** specifica il nome della metrica;
- **Descrizione:** breve descrizione della funzionalità della metrica adottata;

3.6.7 Criteri di accettazione

Per ciascuna metrica elencata nel documento Piano di Qualififca vengono definiti in formato tabellare:

- **Valore di accettazione:** valore che la metrica deve raggiungere per essere considerata soddisfacente o conforme agli standard stabiliti;
- **Valore preferibile:** valore ideale che dovrebbe essere assunto dalla metrica.

3.6.8 Metriche

Metrica	Nome
M1PMS	Percentuale di Metriche Soddisfatte (PMS)
M24DE	Densità degli Errori (DE)

Table 2: Metriche relative alla gestione dei processi

4 *Processi$_G$* organizzativi

Lo sviluppo di software è un campo articolato e multidisciplinare che necessita di una pianificazione meticolosa, una gestione ottimale delle risorse e un controllo stringente della qualità. L'implementazione di processi organizzativi ben definiti diventa quindi essenziale per assicurare il successo del progetto in questione. È fondamentale adottare metodologie strutturate e strumenti adeguati per coordinare le attività, monitorare i progressi e garantire che ogni fase del progetto rispetti gli standard di qualità previsti. Solo attraverso un'attenta pianificazione e una gestione efficace delle risorse è possibile raggiungere gli obiettivi prefissati e consegnare un prodotto finale che soddisfi le aspettative degli utenti.

4.1 Gestione dei *Processi$_G$*

4.1.1 Introduzione

La Gestione dei *Processi$_G$* si dedica a definire, attuare e ottimizzare i processi che orientano lo sviluppo del software, con l'obiettivo di raggiungere i traguardi stabiliti e rispondere alle aspettative degli

stakeholder. Questo ambito include la creazione di procedure ben strutturate, il monitoraggio continuo delle attività e l'adozione di miglioramenti per garantire l'efficienza e la qualità del prodotto finale. È essenziale che ogni fase del processo sia attentamente pianificata e gestita per assicurare che le esigenze degli utenti e degli stakeholder siano pienamente soddisfatte.

4.1.1.1 *Attività$_G$* di gestione di processo

Le attività di Gestione dei *Processi$_G$* includono:

1. **Definizione dei *Processi$_G$*:**
    - Riconoscere e registrare i processi chiave coinvolti nello sviluppo del software.
    - Stabilire direttive e procedure per l'implementazione di ogni processo.
2. **Programmazione e Supervisione:**
    - Creare piani dettagliati per l'implementazione dei processi.
    - Sorvegliare costantemente il progresso, l'efficacia e la conformità ai requisiti stabiliti.
    - Valutare tempi, risorse e costi necessari.
3. **Analisi e Ottimizzazione costante:**
    - Effettuare valutazioni periodiche dei processi per individuare aree di miglioramento.
    - Attuare azioni di correzione e di preventivaggio per ottimizzare i processi.
4. **Formazione e competenze:**
    - Garantire che il personale coinvolto nei processi sia adeguatamente istruito.
    - Conservare e potenziare le competenze indispensabili per una gestione efficiente dei processi.
5. **Gestione dei rischi:**
    - Riconoscere e analizzare i rischi legati ai processi.
    - Stabilire piani per ridurre o affrontare i rischi individuati.

4.1.2 Pianificazione

4.1.2.1 Descrizione

La pianificazione è cruciale nella gestione dei processi, poiché si propone di sviluppare un piano strutturato e coerente per garantire un'efficace realizzazione delle attività lungo tutto il ciclo di vita del software.
Il responsabile del gruppo ha il compito fondamentale di orchestrare ogni aspetto riguardante la pianificazione delle attività, che comprende l'assegnazione delle risorse, la determinazione delle tempistiche e la stesura di piani dettagliati. Inoltre, il responsabile deve assicurarsi che ciascun membro del team possa eseguire con piena fattibilità ed efficienza il piano elaborato da lui stesso.
Di rilevante importanza è la presenza delle risorse necessarie, in termini di tempo, tecnologie e personale, per lo svolgimento di ciascuna attività.

4.1.2.2 Obiettivi

La pianificazione ha come ruolo principale la rotazione dei ruoli all'interno del gruppo, garantendo così che ciascun membro possa assumere, almeno una volta, ciascun ruolo nell'arco della durata del progetto. Lo scopo di tale obiettivo è quello di distribuire equa responsabilità all'interno del team, in modo tale che ciascun membro possa avere una diversa prospettiva dello stesso progetto, conseguendo così un aumento personale delle competenze, necessario per un avanzamento lineare e corretto.

4.1.2.3 Assegnazione dei ruoli

Nell'arco di svolgimento del progetto i ruoli che ciascun membro del gruppo assumerà, sono i seguenti:

1. Responsabile
2. Amministratore
3. Analista
4. Progettista
5. *Verificatore*$_G$
6. *Programmatore*$_G$

4.1.2.4 Responsabile

Persona chiave nell'avanzamento del progetto in quanto coordina il lavoro del gruppo, fungendo come mediatore tra la committente ed il gruppo stesso.
Nello specifico, si occupa di programmare le attività stabilendo quali svolgere, le tempistiche, e l'assegnazione delle priorità. Inoltre, valuta i rischi delle decisioni da prendere, monitora i progressi del progetto, gestisce il personale e approva la documentazione.

4.1.2.5 Amministratore

L'amministratore è una figura professionale, il cui obiettivo è quello di supervisionare e gestire l'ambiente di lavoro utilizzato dal gruppo e anche quello di occuparsi della documentazione relativa alle norme di progetto.

4.1.2.6 Analista

Figura professionale di rilevante importanza all'interno del gruppo, in quanto capace di applicare uno studio approfondito alle esigenze e specifiche del progetto, reinterpretandole in requisiti precisi e completi, indispensabili per tracciare le linee guida utili per un corretto avanzamento del progetto.

4.1.2.7 Progettista

Il progettista è una figura professionale che svolge un ruolo cruciale all'interno di un team di sviluppo software. Esso si occupa di tutte le scelte progettuali derivati dalla stesura dei requisiti applicando, quindi, tecnologie e scelte architetturali più inerenti alla casistica espressa dal lavoro dell'analista.
Il progettista non si occupa della manutenzione del prodotto.

4.1.2.8 *Programmatore*$_G$

Il programmatore è quella figura professionale adibita alla stesura del codice software. Questa figura deve implementare il codice seguendo i requisiti previsti dall'analista ed anche l'architettura idealizzata dal progettista.
Inoltre, il programmatore ha il compito di verificare e validare il codice scritto da se stesso, garantendo qualità e correttezza del prodotto finale.

4.1.2.9 *Verificatore*$_G$

Lo scopo di questa figura professionale è quello di prendere atto del lavoro svolto dagli altri membri del gruppo per constatarne l'effettiva correttezza relativa alla qualità e alla conformità delle attese prefissate. Esso deve, quindi, verificare la conformità dei prodotti alle Norme di *Progetto*$_G$ ed inoltre, verificare la conformità dei prodotti ai requisti funzionali e di qualità stilati dagli analisti, segnalando tutti gli eventuali errori riscontrati nell'attività di verifica.

4.1.2.10 Ticketing

Per tracciare i diversi punti di avanzamento del progetto viene utilizzato GitHub, un sistema appositamente ideato per il tracciamento delle issue, dando così la possibilità di una gestione pratica e chiara delle diverse attività da completare.
La gestione delle issue viene delegata prettamente all'amministratore del gruppo, che in base alle attività rilevate dal responsabile, le crea e le assegna ai membri del gruppo.
Dato l'utilizzo di questo sistema di tracciamento, ogni membro del gruppo, grazie alla presenza di strumenti specifici come la DashBoard e la Roadmap, ha la garanzia di avere una visione completa dello stato di avanzamento delle diverse issue.

Iter di generazione delle issue
Le issue vengono gestite interamente dall'amminstratore del gruppo, il quale deve provvedere a corredarle di specifici attributi, quali:

- **Titolo:** deve essere sintetico e chiaro.
- **Descrizione:** breve e coincisa descrizione della issue.
- **Incaricato:** incaricato allo svolgimento delle issue. Possono esserci più incaricati per issue.
- **Labels:** tag utilizzato per rendere nota la categoria di apparteneza della issue. I label utilizzati sono:
  - **NP:** Norme di *Progetto*$_G$.
  - **PP:** Piano di *Progetto*$_G$.
  - **PQ:** Piano di Qualifica.
  - **AR:** Analisi dei Requisiti.
  - **Poc:** Proof of concept.
- ***Milestone*$_G$:** milestone associata alla issue.
- **Projects**: progetto a cui la issue è associata.
  La presenza di una DashBoard associata ad un progetto, denota la presentazione, al suo interno, delle diverse issue associate a tale progetto.
- **Development:** branch e Pull Request associate alla issue.
  All'accetazione di una Pull Request ne consegue la chiusura della relativa issue ed il suo spostamento nella colonna "Done" della DashBoard.

Ciclo di vita di una issue
Durante il suo ciclo di vita, una issue si sofferma, per un limitato periodo di tempo, in ciascun possibile stato che una issue può assumere. Questi stati sono:

- **Backlog:** in questa fase, le issue vengono create dall'amministratore e raccolte. Le issue nel "Backlog" rappresentano attività che devono essere affrontate ma non ancora pronte per essere lavorate.
- **Ready:** una volta che una issue è stata valutata e prioritizzata, viene posta nella fase "Ready". Questo significa che laissue è pronta per essere iniziata dal membro/i del team incaricato/i.
- **In Progress:** quando l'incaricato inizia a lavorare sulla issue assegnata, questa viene posta nella fase "In Progress". In questa fase, la issue è effettivamente in lavorazione e il suo stato di avanzamento può essere monitorato.
- **In Review:** una volta che la issue viene considerata terminata, l'incaricato apre una Pull Request su GitHub e all'interno della DashBoard, la issue deve essere spostata nella fase "In Review", pronta per essere verificata da un verificatore.
- **Done:** se la issue supera la fase di verifica realizzata dal verificatore, allora essa viene trasferita dalla colonna "In Review" alla colonna "Done", e nel caso di associazione della issue a una Pull Request, una volta che quest'ultima viene confermata dal verificatore, la issue viene automaticamente chiusa e spostata nella colonna "Done" della DashBoard

4.1.2.11 Strumenti

- **GitHub:** sistema utilizzato per il tracciamento e la gestione dell'intero ciclo di vita delle issue.

4.1.3 Coordinamento

4.1.3.1 Descrizione

Con coordinamento viene intesa l'attività che supervisiona la gestione della comunicazione, sia interna, ovvero tra i membri del gruppo, sia esterna, ovvero con il proponente e i committenti, in modo tale che tutte le parti che influiscono sullo sviluppo del progetto possano essere partecipi al suo avanzamento.

4.1.3.2 Obiettivi

L'obiettivo principale del coordinamento è assicurare efficienza nello sviluppo del prodotto software, prevenendo potenziali problemi quali ritardi e malintesi, e concedendo la possibilità a tutti coloro che attivamente partecipano all'avanzamento del progetto, di avere una visione completa e precisa di esso. Una chiara conseguenza del coordinamento è un rafforzamento della coesione dei membri del gruppo, i quali attraverso uno scambio di pareri e punti di vista differenti, possono essere in grado di risolvere agevolmente le diverse problematiche che possono insorgere durante l'avanzamento.

Comunicazione
Per quanto riguarda la comunicazione, il team Byte Your Dreams mantiene comunicazioni costanti, sia interne che esterne del gruppo, che possono essere sincrone o asincrone, in base alle esigenze.

4.1.3.3 Comunicazioni sincrone

- **Comunicazioni sincrone interne**
  Per quanto riguarda le comunicazioni sincrone interne, il gruppo Byte Your Dreams ha deciso di utilizzare *Discord*$_G$ dato che questo servizio è gratuito e permette di comunicare sia attraverso chiamate vocali, videochiamate, messaggi di testo, media, file in chat private, oltre che consentire una comoda utilità di condivisione schermo.
- **Comunicazioni sincrone esterne**
  Per quanto riguarda invece le comunicazioni sincrone esterne è stato scelto, in comune accordo con l'azienda, di utilizzare Teams.

4.1.3.4 Comunicazioni asincrone

- **Comunicazioni asincrone interne**
  Per le comunicazioni asincrone interne viene utilizzata l'applicazione WhatsApp all'interno di un gruppo dedicato.
- **Comunicazioni asincrone esterne**
  Per le comunicazioni asincrone esterne, è stata presa la decisione, in concomitanza con l'azienda, di utilizzare *Gmail*$_G$, un servizio di posta elettronica gratuito offerto da Google oppure il servizio di messaggistica messo a disposizione da Teams.

Riunioni In ogni riunione tenuta dal gruppo viene scelto un segretario con il compito di prendere appunti, attraverso i quali, successivamente, redigere un verbale che contenga tutti i punti chiave trattati durante il meeting.

4.1.3.5 Riunioni interne

È stato scelto di tenere riunioni interne con frequenza settimanale, allo lo scopo di aggiornare tutti i componenti del team sugli avanzamenti avvenuti durante l'arco settimanale dando al team una visione complessiva dello stato del progetto.
Solitamente le riunioni interne avvengono il lunedì o il martedì, in base agli impegni di ciascun membro del gruppo; la maggior parte delle riunioni verranno tenute nella fascia pomeridiana, in quanto la mattina verrà dedicata alle lezioni universitarie.

In caso di assenza di uno dei membri del gruppo, la riunione può essere riprogrammata a quando la disponibilità dei componenti è completa; il tutto avviene tramite una comunicazione interna dei membri del gruppo tramite canale WhatsApp.
Ciascun membro del gruppo ha la libertà di richiedere una riunione ulteriore ogniqualvolta sia necessario; tale riunione verrà organizzata tramite comunicazione interna attraverso WhatsApp.
Le riunioni interne ricoprono un ruolo di elevata importanza per lo svolgimento del progetto in quanto permettono a ciascun membro del gruppo di monitorare gli avanzamenti di ciascuna attività avendo così la possibilità di avere una panoramica completa di esso. Non in secondo luogo, le riunioni interne hanno lo scopo di favorire il dialogo e lo scambio di idee all'interno del team con l'obbiettivo di facilitare la risoluzione di possibili problemi e per la costruzione di un ambiente sereno e favorevole alla comunicazione e alla condivisione.
Le riunioni interne vengono tenute attraverso il canale di comunicazione *Discord*$_G$ per facilitare la comunicazione interna sincrona.
Nelle riunioni interne, il responsabile del gruppo svolge un ruolo di rilevante importanza in quanto:

- Prepara in anticipo gli argomenti da trattare durante la riunione.
- Dà le linee guida delle conversazioni e raccoglie i pareri e punti di vista di ciascun membro.
- Sceglie preventivamente un segretario per la raccolta degli appunti durante lo svolgimento della riunione.
- Al termine della riunione ha il compito di proporre e assegnare le nuove attività da svolgere in prospettiva della futura riunione.

Verbali interni
Le riunioni interne sono così strutturate:

- **Parte iniziale:** viene fatta mente locale sul periodo trascorso dall'ultima riunione interna, cercando di elencare tutti i possibili punti problematici riscontrati e dando, a tutti i membri, lo stato di avanzamento di tutte le precedenti attività.
- **Parte centrale:** vengono discussi tutti i punti stilati nell'ordine del giorno.
- **Parte finale:** vengono proposte le nuove attività da svolgere.

Alla conclusione di ogni riunione, l'amministratore associa una issue in GitHub, relativa all'incarico di redigere il verbale interno, al segretario scelto precedentemente alla riunione.

4.1.3.6 Riunioni esterne

Le riunioni esterne sono essenziali per una fluida continuazione del progetto, in quanto un incontro esterno con l'azienda può risolvere possibili dubbi da parte dei membri.
Il responsabile svolge un ruolo di rilevanza poichè ha il compito di programmare tali incontri in concomitanza con l'azienda e di occuparsi del loro svolgimento garantendo efficienza ed efficacia.
Durante le riunioni esterne sarà il responsabile ad esporre i punti di discussione al proponente, assicurando così una comunicazione ottimale fra le diverse parti e gestendo efficacemente il tempo a disposizione.
I membri del gruppo si sforzano il più possibile per garantire la loro partecipazione agli incontri. Nel caso di impossibilità, dovranno avvertire il prima possibile il responsabile, il quale, a sua volta, provvederà ad aggiornare l'azienda sulla situazione, richiedendo uno slittamento dell'incontro.

Riunione con l'azienda proponente
In concomitanza con l'azienda, si è deciso di tenere un *SAL*$_G$ (stato di avanzamento lavori) con frequenza bisettimanale tramite il canale di comunicazione Teams.
La scaletta della riunione esterna è la seguente:

- **Revisione del periodo precedente:** viene fatto il punto della situazione su quanto fatto dall'ultimo *SAL*$_G$ mettendo in evidenza i punti risolti ed i punti non risolti.

- **Ordine del giorno:** principali punti di discussione della riunione, di particolare interesse per i membri del gruppo e indispensabili per l'avanzamento del progetto.
- **Chiarimenti ulteriori:** ulteriori punti di discussione che solitamente conseguono punti di discussione relativi all' "Ordine del giorno". Dubbi che derivano da altri dubbi maggiori oppure dubbi di interesse secondario.

Verbali esterni
I verbali esterni vengono redatti dal segretario e successivamente, il responsabile si occupa della condivisione di esso con l'azienda proponente tramite Google Drive.

4.1.3.7 Strumenti

- ***Discord*$_G$:** utilizzato per la comunicazione sincrona e per le riunioni interne.
- **WhatsApp:** utilizzato per la comunicazione asincorona con i membri del gruppo.
- **Teams:** utilizzato per le riunioni esterne e per la comunicazione asincora con l'azienda proponente.
- ***Gmail*$_G$:** servizio di posta elettronica utilizzato per la comunicazione asincrona con l'azienda.
- **Google Drive:** servizio di condivisione di documenti messo a disposizione da Google ed utilizzato dal gruppo per la condivisione di verbali esterni con l'azienda proponente.

4.1.3.8 Metriche

Metrica	Nome
M6SV	Schedule Variance (SV)
M21IF	Implementazione delle Funzionalità (IF)
M4BV	Budget Variance (BV)
M12VR	Variazione dei Requisiti (VR)

Table 3: Metriche relative alla gestione dei processi

5 *Standard*$_G$ per la qualità

Durante la valutazione della qualità dei processi e del software si è fatto affidamento a degli standard internazionali sviluppati per fornire una struttura coerente e standardizzata per identificare, misurare e valutare la qualità di un prodotto software. Lo standard ISO/IEC 9126 viene usato per gestire parametri come funzionalità, affidabilità, usabilità, efficienza, manutenibilità e portabilità. Questa suddivisione garantisce di avere un controllo completo sulla qualità del prodotto software.
Lo standard ISO/IEC 12207:1995 invece suddivide i processi software in primari,di supporto e organizzativi; è progettato per fornire una struttura che descriva le attività e i compiti necessari per sviluppare, utilizzare e mantenere un sistema software.
Infine, lo standard ISO/IEC 25010 fornisce un framework che permette di individuare e misurare le metriche di qualità del software.

5.1 Caratteristiche del *Sistema*$_G$, *Standard*$_G$ ISO/IEC 25010

5.1.1 Funzionalità

- **Adeguatezza funzionale:** Copertura completa di tutte le funzionalità richieste;
- **Accuratezza:** Capacità del software di fornire risultati corretti e precisi;

- ***Interoperabilità$_G$*:** Capacità del software di interagire con altri sistemi.

5.1.2 Affidabilità

- **Maturità:** Capacità del software di evitare errori o guasti in condizioni standard;
- **Tolleranza agli Errori:** Capacità del software di continuare a funzionare anche in presenza di guasti;
- **Recuperabilità:** Capacità del software di ripristinarsi dopo un errore.

5.1.3 Usabilità

- **Comprensibilità:** Facilità con cui gli utenti possono comprendere il software;
- **Apprendibilità:** Facilità con cui un utente può imparare a utilizzare il software;
- **Operabilità:** Facilità d'uso durante l'operatività del software;

5.1.4 Efficienza

- **Tempo di Risposta:** Tempo impiegato dal software per rispondere alle richieste dell'utente;
- **Utilizzo delle Risorse:** Efficienza nell'uso delle risorse del sistema.
- **Capacità:** Capacità del software di gestire quantità crescenti di utenti o dati.

5.1.5 Manutenibilità

- **Analizzabilità:** Facilità con cui è possibile diagnosticare guasti o identificare le parti da modificare;
- **Modificabilità:** Capacità di apportare modifiche al software senza difficoltà;
- **Stabilità:** Capacità del software di funzionare correttamente dopo le modifiche;

5.1.6 Portabilità

- **Adattabilità:** Capacità del software di essere adattato a diversi ambienti senza modifiche estese;
- **Installabilità:** Facilità con cui il software può essere installato in un ambiente specifico;
- **Conformità:** il rispetto delle norme e degli standard relativi alla portabilità.

5.2 Suddivisione secondo *Standard*$_G$ ISO/IEC 12207:1995

5.2.1 *Processi*$_G$ primari

Sono gli elementi fondamentali legati al ciclo di vita del software, che includono lo sviluppo, l'implementazione e la gestione del software. Questi processi sono al centro del modello e sono progettati per guidare le organizzazioni nella creazione, acquisizione e manutenzione dei sistemi software.

5.2.2 *Processi*$_G$ di supporto

I processi di supporto forniscono strumenti e attività per assistere i processi primari e organizzativi, contribuendo a garantire la qualità del software e il corretto svolgimento delle attività. Questi processi non producono direttamente il software, ma sono fondamentali per assicurare che il prodotto finale soddisfi i requisiti e sia conforme agli standard.

5.2.3 *Processi*$_G$ organizzativi

I processi organizzativi sono progettati per supportare la gestione complessiva del ciclo di vita del software a livello aziendale e assicurare che le attività siano pianificate, monitorate e migliorate in modo sistematico. Questi processi si concentrano sugli aspetti strategici e gestionali, fornendo un quadro di riferimento per una continua ottimizzazione.

6 Metriche di qualità

Le metriche di qualità di un prodotto software sono strumenti di misurazione impiegati per analizzare e monitorare vari aspetti della qualità del software e del processo di sviluppo. Il loro scopo è assicurare che il software rispetti i requisiti funzionali e non funzionali evidenziando eventuali aree che necessitano miglioramenti e garantire il rispetto alle conformità dello stesso.

6.1 Metriche per la qualità di processo

- **Metrica M1PMS:**
  - **Nome:** Percentuale di Metriche Soddisfatte (PMS);
  - **Descrizione:** Misura la proporzione delle metriche definite che sono state effettivamente adottate o soddisfatte in un progetto;
  - **Formula:** $\frac{metriche\ soddisfatte}{metriche\ totali} \times 100$
- **Metrica M2EAC:**
  - **Nome:** Estimated at Completion (*EAC*$_G$);
  - **Descrizione:** Fornisce una stima del costo totale previsto per completare un progetto;
  - **Formula:** $EAC_G = \frac{BAC}{CPI_G}$
- **Metrica M3CPI:**
  - **Nome:** Cost *Performance*$_G$ Index (*CPI*$_G$);
  - **Descrizione:** Misura l'efficienza dei costi di un progetto;
  - **Formula:** $CPI_G = \frac{EV}{AC}$
- **Metrica M4BV:**

  - **Nome:** Budget Variance (BV);
  - **Descrizione:** Calcola la differenza percentuale tra il budget pianificato e l'effettivo costo sostenuto;
  - **Formula:** $BV = EV - AC$
- **Metrica M5AC:**
  - **Nome:** Actual Cost (AC);
  - **Descrizione:** Misurazione aggiornata dei costi effettivamente sostenuti dall'inizio di un progetto;
  - **Formula:** Dato disponibile e aggiornato in "Piano di progetto" per ogni periodo.
- **Metrica M6SV:**
  - **Nome:** Schedule Variance (SV);
  - **Descrizione:** Misura quanto un progetto è in anticipo o in ritardo rispetto alla pianificazione temporale;
  - **Formula:** $SV = EV - PV_G$
- **Metrica M7EV:**
  - **Nome:** Earned Value (EV);
  - **Descrizione:** Misura il valore del lavoro effettivamente completato fino a una data specifica;
  - **Formula:** $EV = \%lavoro\ svolto \times EAC_G$
- **Metrica M8PV:**
  - **Nome:** Planned Value (PV_G);
  - **Descrizione:** Fornisce una stima dei costi previsti per le stakeholder future di un progetto;
  - **Formula:** $PV_G = \%lavoro\ svolto \times BAC$
- **Metrica M9ETC:**
  - **Nome:** Estimate to Complete (ETC_G);
  - **Descrizione:** Fornisce una stima dei costi necessari per completare un progetto, basandosi sul lavoro rimanente e sulle spese previste;
  - **Formula:** $ETC_G = EAC_G - AC$
- **Metrica M11RNP:**
  - **Nome:** Rischi Non Previsti (RNP);
  - **Descrizione:** Fornisce una misurazione della capacità di un progetto di evitare l'emergere di rischi non identificati in fase di pianificazione;
- **Metrica M12VR:**
  - **Nome:** Variazione dei Requisiti (VR);
  - **Descrizione:** Metrica della stabilità dei requisiti di un progetto, indicando quanto sono stati soggetti a modifiche rispetto alla pianificazione iniziale;
  - **Formula:** $NRA + NRR + NRM$, dove:
    * NRA (Numero Requisiti Aggiunti) è la quantità di requisiti aggiunti dall'ultimo incremento;
    * NRR (Numero Requisiti Rimossi) è la quantità di requisiti rimossi dall'ultimo incremento;
    * NRM (Numero Requisiti Modificati) è la quantità di requisiti modificati dall'ultimo incremento.

- **Metrica M13PCTS:**
  - **Nome:** Percentuale di Casi di *Test*$_G$ Superati (PCTS);
  - **Descrizione:** Fornisce una misura della qualità del software, indicando quanto efficacemente il software ha superato i test di verifica;
  - **Formula:** $\frac{numero\ di\ casi\ di\ test\ superati}{numero\ totale\ di\ casi\ di\ test} \times 100$
- **Metrica M14PCTF:**
  - **Nome:** Percentuale di Casi di *Test*$_G$ Falliti (PCTF);
  - **Descrizione:** Fornisce una misura della qualità del software, indicando quanti test non sono stati superati durante la fase di verifica;
  - **Formula:** $\frac{numero\ di\ casi\ di\ test\ non\ superati}{numero\ totale\ di\ casi\ di\ test} \times 100$
- **Metrica M15SC:**
  - **Nome:** Statement Coverage (*MSC*$_G$);
  - **Descrizione:** Misura la percentuale di istruzioni eseguite almeno una volta durante i test;
  - **Formula:** $MSC_G = \frac{istruzioni\ eseguite}{istruzioni\ totali} \times 100$
- **Metrica M16BC:**
  - **Nome:** *Branch*$_G$ Coverage (MBC);
  - **Descrizione:** Misura la percentuale di rami eseguiti nei punti di decisione del codice;
  - **Formula:** $MBC = \frac{flussi\ funzionali\ implementati\ e\ testati}{flussi\ condizionali\ riusciti\ e\ non} \times 100$
- **Metrica M17CNC:**
  - **Nome:** Condition Coverage (CNC);
  - **Descrizione:** Misura la percentuale di condizioni booleane valutate sia come true sia come false all'interno di ciascun punto di decisione;
  - **Formula:** $CNC = \frac{numero\ di\ operandi\ eseguiti}{numero\ totale\ di\ operandi\ eseguiti} \times 100$

6.2 Metriche per la qualità di prodotto

- **Metrica M18PROS:**
  - **Nome:** Percentuale di Requisiti Obbligatori Soddisfatti (PROS);
  - **Descrizione:** Metrica cruciale per valutare il successo di un progetto software, verifica la conformità del prodotto ai requisiti essenziali definiti nell'analisi dei requisiti;
  - **Formula:** $\frac{requisiti\ obbligatori\ soddisfatti}{requisiti\ obbligatori\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M19PRDS:**
  - **Nome:** Percentuale di Requisiti Desiderati Soddisfatti (PRDS);
  - **Descrizione:** Misura l'estensione con cui i requisiti che migliorano l'esperienza dell'utente sono stati implementati, contribuendo a creare un prodotto non solo funzionale ma anche più piacevole e completo;
  - **Formula:** $\frac{requisiti\ desiderabili\ soddisfatti}{requisiti\ desiderabili\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M20PRPS:**

  - **Nome:** Percentuale di Requisiti Opzionali Soddisfatti (PRPS);
  - **Descrizione:** Metrica che valuta la quantità di requisiti opzionali implementati in un prodotto software;
  - **Formula:** $\frac{requisiti\ opzionali\ soddisfatti}{requisiti\ opzionali\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M21IF:**
  - **Nome:** Implementazione delle Funzionalità (IF);
  - **Descrizione:** Indicatore diretto del grado di completamento dello sviluppo software in relazione alle funzionalità inizialmente pianificate;
  - **Formula:**$(1 - \frac{F_{NL}}{F_L}) \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M22CO:**
  - **Nome:** Correttezza Ortografica (CO);
  - **Descrizione:** Valuta l'assenza di errori di ortografia nella documentazione del software;
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M23IG:**
  - **Nome:** Indice Gulpease (MIG);
  - **Descrizione:** Valuta la leggibilità di un testo in base alla lunghezza media delle parole e delle frasi;
  - **Formula:** $IG = 89 + \frac{300 \cdot N_f - 10 \cdot N_l}{N_p}$
  - **Dove:**
    - N_f: numero di frasi;
    - N_l: numero di lettere;
    - N_p: numero di parole.
  - I risultati sono compresi tra 0 e 100, dove il valore "100" indica la leggibilità più alta e "0" la leggibilità più bassa. In generale risulta che i testi con un indice:
    - < 80: sono difficili da leggere per chi ha la licenza elementare;
    - < 60: sono difficili da leggere per chi ha la licenza media;
    - < 40: sono difficili da leggere per chi ha un diploma superiore.
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M24DE:**
  - **Nome:** Densità degli Errori (DE);
  - **Descrizione:** Metrica che calcola la percentuale di errori presenti nel codice di un software rispetto alla quantità totale di codice;
  - **Formula:** $DE = \frac{numero\ di\ errori}{totale\ delle\ linee\ di\ codice} \times 100$
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M25ATC:**
  - **Nome:** Accoppiamento tra Classi (ATC);
  - **Descrizione:** Misura il grado di dipendenza e interconnessione tra le classi di un sistema software;
  - **Caratteristica di qualità:** Manutenibilità.

- **Metrica M26MCCM**
  - **Nome:** Complessità Ciclomatica per Metodo ($MCCM_G$);
  - **Descrizione:** Misura la complessità di un metodo in base ai possibili livelli di annidamento percorribili all'interno di esso;
  - **Formula:** $MCCM_G = e - n + 2$
  - **Dove:**
    - e: Numero di archi del grafo del flusso di esecuzione del metodo;
    - n: Numero di vertici del grafo del flusso di esecuzione del metodo.
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M27PM:**
  - **Nome:** Parametri per Metodo (PM);
  - **Descrizione:** Valore massimo di parametri per ogni metodo;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M28APC:**
  - **Nome:** Attributi Per Classe (APC);
  - **Descrizione:** Calcola l'ammontare massimo di attributi per classe;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M29LCM:**
  - **Nome:** Linee di Codice per Metodo (LCM);
  - **Descrizione:** Misura la lunghezza massima di ogni metodo dalla lunghezza del codice (in linee);
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M30PG:**
  - **Nome:** Profondità delle Gerarchie (PG);
  - **Descrizione:** Misura il numero di livelli tra una classe base e le sue sottoclassi in una gerarchia di ereditarietà;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M31TMR:**
  - **Nome:** Tempo Medio di Risposta (TMR);
  - **Descrizione:** Valuta quanto un sistema software è veloce e reattivo nel rispondere alle richieste;
  - **Formula:** $\frac{somma\ dei\ tempi\ di\ risposta}{numero\ totale\ di\ misurazioni}$
  - **Caratteristica di qualità:** Efficienza.
- **Metrica M32FU:**
  - **Nome:** Facilità di Utilizzo (FU);
  - **Descrizione:** Metrica che valuta il numero di click necessari per raggiungere un obiettivo all'interno del sistema software;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M33TA:**

  - **Nome:** Tempo di Apprendimento (TA);
  - **Descrizione:** Misura il tempo massimo necessario a un utente per imparare ad usare un prodotto;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M34VBS:**
  - **Nome:** Versioni dei $Browser_G$ Supportate (VBS);
  - **Descrizione:** Calcola in percentuale quante sono le versioni di browser compatibili con un sistema software rispetto al totale delle versioni disponibili;
  - **Caratteristica di qualità:** Portabilità.