ByteOps.swe@gmail.com

Norme di progetto

Informazioni documento

Redattori	A. Barutta R. Smanio N. Preto F. Pozza
Verificatori	E. Hysa L. Skenderi D. Diotto
Destinatari	T. Vardanega R. Cardin

Registro delle modifiche

Versione	Data	Autore	Verificatore	Dettaglio
2.0.0	23/03/2024	R. Smanio	A. Barutta	Correzzione sez. test e revisione finale
1.1.0	21/03/2024	R. Smanio	A. Barutta	Completata sezione Continuous Integration
1.0.0	14/02/2024	F. Pozza	R. Smanio	Correzione errori grammaticali e di formattazione.
0.7.0	06/01/2024	R. Smanio	L. Skenderi	Aggiornamento sezione Gestione della configurazione.
0.6.0	27/12/2023	D. Diotto	N. Preto	Completata sezione Metriche di qualità.
0.5.1	22/12/2023	R. Smanio	A. Barutta	Iniziale stesura sezione Metriche di qualità.
0.5.0	18/12/2023	R. Smanio	D. Diotto	Completata sezione Standard per la qualità.
0.4.1	10/12/2023	R. Smanio	N. Preto	Prima stesura sottosezioni di Standard per la qualità.
0.4.0	05/12/2023	L. Skenderi	D. Diotto	Completate sottosezioni di Processi organizzativi.
0.3.2	04/12/2023	R. Smanio E. Hysa	D. Diotto	Inizio stesura Formazione e aggiornamento Gestione dei Processi.
0.3.1	02/12/2023	L. Skenderi E. Hysa	D. Diotto	Inizio stesura Miglioramento e Gestione dei Processi.
0.3.0	01/12/2023	L. Skenderi E. Hysa	R. Smanio	Completate sottosezioni sezione Processi di supporto.
0.2.3	24/11/2023	L. Skenderi R. Smanio	D.Diotto	Aggiornate sezioni Gestione della configurazione e Gestione della qualità.

Versione	Data	Autore	Verificatore	Dettaglio
0.2.2	22/11/2023	F.Pozza R. Smanio	E. Hysa	Iniziale stesura Risoluzione dei problemi delle configurazioni e Gestione della qualità.
0.2.1	20/11/2023	A. Barutta R. Smanio	E. Hysa	Iniziale stesura Gestione delle configurazioni e Joint Review.
0.2.0	15/11/2023	F. Pozza R. Smanio	E. Hysa	Completate Fornitura, Sviluppo, Verifica e Validazione.
0.1.2	10/11/2023	F. Pozza A. Barutta	E. Hysa	Aggiornate Fornitura, Sviluppo e inziata sezione Verifica.
0.1.1	07/11/2023	F. Pozza R. Smanio	E. Hysa	Inzio stesura Fornitura e Sviluppo.
0.1.0	05/11/2023	F. Pozza R. Smanio	E. Hysa	Sezione Documentazione, Introduzione.

Indice

ByteOps

Contents

List of Figures

1 Introduzione

1.1 Finalità del documento

L'obiettivo fondamentale del seguente documento è quello di stabilire le linee guida e le *best practice* che ciascun membro del gruppo deve seguire per garantire un approccio efficiente ed efficace nel processo di realizzazione del progetto didattico.
I *processi*$_G$ e le relative *attività*$_G$ contenute nel seguente documento sono state definite a partire dallo Standard ISO/IEC 12207:1995.

Il documento è strutturato secondo i *processi*$_G$ del ciclo di vita del *software*$_G$ e presenta una gerarchia in cui ogni processo si configura come una serie di *attività*$_G$. Ciascuna *attività*$_G$ è composta da procedure dotate di obiettivi, scopi e strumenti ben definiti.
In aggiunta, il documento dettaglia le convenzioni relative all'utilizzo dei diversi strumenti adottati durante lo sviluppo del prodotto.

È importante sottolineare che questo documento è in continua evoluzione poiché le norme definite al suo interno vengono regolarmente riesaminate, aggiornate e ottimizzate seguendo un approccio incrementale.

1.2 Glossario

Nella documentazione è incluso il ***Glossario***, dove vengono definiti tutti i termini specifici o potenzialmente ambigui presenti nei vari documenti correlati al progetto. La presenza di una nota a pedice con la lettera *G* accanto a un termine indica che è possibile trovare la sua definizione nel ***Glossario***.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- **Standard ISO/IEC 12207:1995**
  Standard internazionale che definisce un modello di ciclo di vita del *software*$_G$ e in cui sono definite delle linee guida per la gestione dei *processi*$_G$ *software*$_G$ e le relative *attività*$_G$.

  In sintesi, stabilisce una struttura per organizzare le diverse fasi e le *attività*$_G$ coinvolte nel ciclo di vita del *software*$_G$, aiutando le organizzazioni a sviluppare prodotti *software*$_G$ in modo più efficiente ed efficace.

- **Capitolato C6**

1.3.2 Riferimenti informativi

- "Clean Code" di Robert C. Martin;
- **Documentazione Git**;
- **Documentazione LaTeX**;
- **Documentazione Python**;
- **Documentazione Kafka**;
- **Documentazione Clickhouse**;
- **Documentazione Grafana**.

2 Processi primari

2.1 Fornitura

2.1.1 Introduzione

Conformemente allo *standard*$_G$ ISO/IEC 12207:1995, il processo di fornitura definisce un insieme strutturato di *attività*$_G$, metodi, pratiche e procedure mirate a garantire la fornitura del prodotto *software*$_G$ richiesto dal *committente*$_G$. In particolare, il processo di fornitura si concentra sul monitoraggio e sul coordinamento delle *attività*$_G$ eseguite dal team durante la realizzazione del progetto, dalla concezione alla consegna, assicurando che il prodotto finale soddisfi i requisiti specificati dal *committente*$_G$ e venga consegnato nei tempi e nei costi previsti.

Tale processo viene attuato una volta completata la redazione integrale del documento *Valutazione Capitolati*, cioè dopo aver correttamente identificato le specifiche e i vincoli richiesti dal *proponente*$_G$.

In seguito, il *fornitore*$_G$ dovrà instaurare un contratto con l'azienda *proponente*$_G$, nel quale si concorderanno i requisiti, i vincoli e i tempi di consegna del prodotto finale. Una volta concluso l'accordo con il *proponente*$_G$, sarà possibile iniziare il processo di redazione del documento *Piano di Progetto*, il quale delineerà le *attività*$_G$, le risorse e i costi indispensabili per la realizzazione del prodotto.

2.1.2 Attività

Il processo di fornitura, come descritto dallo *standard*$_G$ ISO/IEC 12207:1995, è composto dalle seguenti *attività*$_G$:

1. **Avvio**: si individuano le necessità e i requisiti del cliente e si avvia l'iter per rispondere alle sue esigenze;
2. **Preparazione della risposta alle richieste**: si elaborano le proposte per rispondere alle richieste del cliente, che comprendono la definizione dei requisiti e delle condizioni contrattuali;
3. **Contrattazione**: questa *attività*$_G$ coinvolge la negoziazione dei termini contrattuali tra il *fornitore*$_G$ e il cliente, incluso il raggiungimento di un accordo sui requisiti del progetto e sulle condizioni di consegna;
4. **Pianificazione**: si pianificano le *attività*$_G$ necessarie per soddisfare i requisiti del cliente, inclusi i tempi, le risorse e i costi;
5. **Esecuzione e controllo**: vengono eseguite le *attività*$_G$ pianificate e viene monitorato il progresso del progetto per garantire il rispetto dei tempi, dei costi e dei requisiti;
6. **Revisione e valutazione**: si effettuano revisioni periodiche per valutare lo stato del progetto rispetto agli obiettivi pianificati e per identificare eventuali problemi o rischi;
7. **Consegna e completamento**: il prodotto *software*$_G$ viene consegnato al cliente includendo la documentazione finale e attuando le ultime procedure e compiti necessari per concludere il progetto in modo completo e soddisfacente.

2.1.3 Comunicazioni con l'azienda proponente

Il gruppo *ByteOps* instaurerà e si impegnerà a mantere una comunicazione costante con l'azienda *proponente*$_G$ *Sync Lab* in modo da ottenere un riscontro sul lavoro svolto fino a quel momento e per verificare che i requisiti individuati siano conformi a quanto stabilito nel capitolato e nei colloqui con la *proponente*$_G$ stessa.

L'azienda *proponente*$_G$ *Sync Lab* mette a disposizione un indirizzo mail per contattare il team *ByteOps* relativamente alle comunicazioni e le richieste formali, ed un canale *Element* per comunicazioni informali, quali richiedere eventuali chiarimenti o richieste di aiuto in determinate sezioni dello svolgimento.

Gli incontri *SAL*$_G$ (*Stato Avanzamento Lavori*) con la *proponente*$_G$ sono fissati a cadenza bisettimanale, in modo da poter discutere con essa lo stato di avanzamento e le difficoltà

riscontrate per quanto concerne gli obiettivi proposti nel periodo che intercorre tra un *SAL*$_G$ e il successivo, nonchè le *attività*$_G$ future da svolgere. Per ogni incontro effettuato con la *proponente*$_G$, verrà redatto un ***Verbale esterno*** che riporterà data e ora dell'incontro, i partecipanti, gli argomenti trattati e le *attività*$_G$ concordate per il prossimo incontro. I verbali esterni sono archiviati al percorso *Nome_periodo/Verbali/Esterni* nella repository del gruppo *ByteOps*.

2.1.4 Documentazione fornita

Viene elencata di seguito la documentazione che il gruppo *ByteOps* consegnerà ai commitenti *Prof.* Tullio Vardanega e *Prof.* Riccardo Cardin e all'azienda *proponente*$_G$ *Sync Lab*.

2.1.4.1 Valutazione dei capitolati

Il documento *Valutazione Capitolati* contiene l'analisi dei capitolati proposti, con l'obiettivo di individuare il capitolato più adatto alle esigenze del gruppo *ByteOps*.
Il documento contiene le seguenti sezioni:

- **Informazioni generali**: contiene informazioni generali sul capitolato, come il nome del progetto e la *proponente*$_G$.
- **Obiettivo**: contiene una sintesi del prodotto da sviluppare seguendo quanto richiesto dal capitolato.
- **Tecnologie suggerite**: contiene le tecnologie suggerite dal *proponente*$_G$ per lo sviluppo del prodotto.
- **Considerazioni**: contiene le considerazioni del gruppo *ByteOps* riguardo il capitolato, come i pro e i contro, le criticità e le motivazioni che hanno portato alla scelta o meno del capitolato.

2.1.4.2 Analisi dei requisiti

L'*Analisi dei Requisiti* è un documento che contiene la descrizione in dettaglio dei requisiti, dei casi d'uso e definisce in modo esaustivo le funzionalità che il prodotto offirà.
Tale documento ha come scopo quello di eliminare ogni ambiguità che potrebbe sorgere durante la lettura del capitolato e di fornire una base di partenza per la progettazione del prodotto.
Il documento contiene le seguenti sezioni:

- **Introduzione**: contiene una breve descrizione del prodotto e delle sue funzionalità.
- **Casi d'uso**: identifica tutti i possibili scenari di utilizzo da parte dell'utente del prodotto. Ogni caso d'uso è accompagnato da una descrizione che ne descrive il funzionamento.

- **Requisiti**: insieme di tutte le richieste e vincoli definiti dalla *proponente*$_G$, o estratti da discussioni con il gruppo, per realizzare il prodotto *software*$_G$ commissionato. Ogni requisito è accompagnato da una descrizione e da un relativo caso d'uso.

2.1.4.3 Piano di progetto

Il *Piano di Progetto* è un documento stilato ed aggiornato continuamente che tratta i seguenti temi:

- **Analisi dei rischi**: contiene l'analisi dei rischi che il gruppo prevede di incontrare durante lo svolgimento del progetto. Ad ogni rischio è associata una descrizione, una procedura di identificazione, la probabilità di occorrenza, l'indice di gravità e un relativo piano di mitigazione del rischio. Tale documento permetterà di attuare le strategie di mitigazione dei rischi in modo tempestivo.
- **Pianificazione e metodo utilizzato**: contiene la pianificazione delle *attività*$_G$, la suddivisione dei ruoli e l'identificazione delle *attività*$_G$ da svolgere per ogni ruolo. In aggiunta, fornisce una dettagliata esposizione del modello di sviluppo adottato, corredato dalle motivazioni sottostanti che hanno orientato la decisione verso tale approccio.
- **Preventivo e consuntivo di periodo**: contiene il preventivo delle ore e dei costi associati a ciascuna *attività*$_G$, suddivisi per ogni periodo definito. Al termine di ciascun periodo, viene redatto il consuntivo, il quale include un resoconto delle ore e dei costi effettivamente sostenuti per ogni singola *attività*$_G$.

2.1.4.4 Piano di qualifica

Il *Piano di Qualifica* è un documento essenziale che illustra le strategie di verifica e validazione per garantire che il prodotto soddisfi le aspettative del *committente*$_G$ e della *proponente*$_G$. Esso identifica le metodologie di verifica, i criteri di validazione e le risorse coinvolte, assicurando un processo di controllo qualità efficiente. Il documento contiene le seguenti sezioni:

- **Qualità di processo**: contiene le strategie di verifica e validazione per garantire che i *processi*$_G$ di sviluppo del prodotto soddisfino gli obiettivi di qualità.
- **Qualità di prodotto**: contiene le strategie di verifica e validazione per garantire che il prodotto soddisfi gli obiettivi di qualità.
- **Specifica dei test**: contiene la descrizione approfondita dei *test*$_G$.

2.1.4.5 Glossario

Il *Glossario* è un documento essenziale finalizzato alla chiarificazione dei termini tecnici e degli acronimi impiegati all'interno dei documenti. La sua funzione principale è fornire agli

utenti e ai lettori un riferimento chiaro e uniforme, garantendo così una interpretazione coerente dei concetti specifici presenti nei diversi documenti. Questo strumento si propone inoltre di mitigare eventuali ambiguità e prevenire fraintendimenti, promuovendo una comprensione univoca dei termini tecnici utilizzati nell'ambito della documentazione.

2.1.4.6 Lettera di presentazione

La *Lettera di presentazione* è un documento che accompagna la presentazione della documentazione e del prodotto *software*$_G$ durante le fasi di revisione di progetto. Questo documento dettaglia l'elenco della documentazione redatta, la quale sarà consegnata ai committenti, il *Prof.* Tullio Vardanega e il *Prof.* Riccardo Cardin, nonché all'azienda *proponente*$_G$, *Sync Lab*.

2.1.5 Strumenti

Gli strumenti utilizzati dal gruppo per la gestione del processo di fornitura sono:

- **Google Meet:** *servizio*$_G$ di videoconferenza utilizzato per gli incontri con la *proponente*$_G$;
- **Google Calendar:** *servizio*$_G$ di *calendario*$_G$ utilizzato per la pianificazione degli incontri con la *proponente*$_G$;
- **Element:** *servizio*$_G$ di messaggistica istantanea utilizzato per le comunicazioni con la *proponente*$_G$ in caso di necessità;
- **Google Slides:** *servizio*$_G$ cloud utilizzato per la creazione delle presentazioni da mostrare durante i diari di bordo;
- **Google Sheets:** *servizio*$_G$ cloud utilizzato per la creazione dei grafici di preventivo e consuntivo di periodo presenti nel documento *Piano di Progetto*.

2.2 Sviluppo

2.2.1 Introduzione

L'ISO/IEC 12207:1995 stabilisce le linee guida per il processo di sviluppo, il quale include *attività*$_G$ cruciali come analisi, progettazione, codifica, *integrazione*$_G$, testing, installazione e accettazione. È fondamentale svolgere queste *attività*$_G$ in stretta aderenza alle linee guida e ai requisiti definiti nel contratto con il cliente, garantendo così un'implementazione accurata e conforme alle specifiche richieste.

2.2.2 Analisi dei requisiti

2.2.2.1 Descrizione

L'*analisi dei requisiti*$_G$ è un' *attività*$_G$ critica nell'ambito dello sviluppo *software*$_G$ poiché stabilisce le basi per il design, l'implementazione e i *test*$_G$ del *sistema*$_G$.
Secondo lo *standard*$_G$ ISO/IEC 12207:1995, lo scopo dell'*analisi dei requisiti*$_G$ è di comprendere e definire in modo esaustivo le esigenze del cliente e del *sistema*$_G$.

L'*attività*$_G$ di analisi richiede di rispondere a domande fondamentali come: "Qual è il dominio?", "Qual è la cosa giusta da fare?", "Quali sono le necessità del cliente?" e consiste nella comprensione approfondita del dominio e nella definizione chiara di obiettivi, vincoli e requisiti sia tecnici che funzionali.

2.2.2.2 Obiettivi

- Definire insieme alla *proponente*$_G$ gli obiettivi del prodotto per rispecchiarne le aspettative, comprendendo identificazione, documentazione e validazione dei requisiti funzionali e non funzionali;
- Facilitare la comprensione comune tra gli *stakeholder*$_G$;
- Permettere una stima sulle tempistiche e sui costi;
- Fornire ai progettisti requisiti chiari e di facile comprensione;
- Favorire l'*attività*$_G$ di verifica e *test*$_G$ fornendo dei riferimenti pratici.

2.2.2.3 Documentazione

È compito degli analisti effettuare l'*analisi dei requisiti*$_G$, redigendo un documento con il medesimo nome che deve contenere:

- **Introduzione**: presentazione e scopo del documento stesso;
- **Descrizione**: analisi del prodotto
  - Obiettivi del prodotto;
  - Funzionalità del prodotto;
  - Caratteristiche utente;
  - Tecnologie.

- **Casi d'uso**: funzionalità offerte dal *sistema*$_G$ dal punto di vista dell'utente
  - Attori: utenti esterni al *sistema*$_G$;
  - Elenco dei casi d'uso:
    - Casi d'uso in formato testuale;
    - Diagrammi dei casi d'uso.
  - Eventuali diagrammi di *attività*$_G$: permettono di facilitare la comprensione dei *processi*$_G$ relativi alle funzionalità.
- **Requisiti**:
  - Requisiti funzionali;
  - Requisiti qualitativi;
  - Requisiti di vincolo.

2.2.2.4 Casi d'uso

Forniscono una descrizione dettagliata delle funzionalità del *sistema*$_G$ dal punto di vista degli utenti, identificando come il *sistema*$_G$ risponde a determinate azioni o scenari. In breve, i casi d'uso sono strumenti utilizzati nell'*analisi dei requisiti*$_G$ per catturare e illustrare in modo chiaro e comprensibile come gli utenti interagiranno con il *software*$_G$ e quali saranno i risultati di tali interazioni.

Ogni caso d'uso testuale deve essere costituito da:

1. **Identificativo**

   UC [Numero caso d'uso] . [Numero sotto caso d'uso] - [Titolo]

   (ex. UC6.1 - Visualizzazione posizione *sensore*$_G$).

   con:

   - **Numero caso d'uso:** *ID*$_G$ numerico del caso d'uso;
   - **Numero sotto caso d'uso:** *ID*$_G$ numerico del sottocaso d'uso (presente esclusivamente se si sta identificando un sottocaso d'uso).
   - **Titolo**: titolo breve ed esplicativo del caso d'uso.

2. **Attore principale:** entità esterna che interagisce attivamente con il *sistema*$_G$ per soddisfare una sua necessità;

3. **Attore secondario:** eventuale entità esterna che non interagisce attivamente con il *sistema*$_G$, ma all'interno di un caso d'uso permette al *sistema*$_G$ di soddisfare il bisogno dell'attore principale;
4. **Descrizione:** eventuale descrizione breve della funzionalità;
5. **Scenario principale:** sequenza di eventi che si verificano quando un attore interagisce con il *sistema*$_G$ per raggiungere l'obbiettivo del caso d'uso (postcondizioni);
6. **Estensioni:** eventuali scenari alternativi che, in seguito ad una o più specifiche condizioni, portano il flusso del caso d'uso a non giungere alle postcondizioni;
7. **Precondizioni:** stato in cui si deve trovare il *sistema*$_G$ affinché la funzionalità sia disponibile all'attore;
8. **Postcondizioni:** stato in cui si trova il *sistema*$_G$ dopo l'esecuzione dello scenario principale;
9. **User story associata:** breve descrizione di una funzionalità del *software*$_G$, scritta dal punto di vista dell'utente, che fornisce contesto, obiettivi e valore.
   L'user story viene scritta nella forma: "Come [utente] desidero poter [funzionalita] per [valore aggiunto]".

2.2.2.5 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono strumenti grafici che permettono di visualizzare in modo chiaro e intuitivo le funzionalità offerte dal *sistema*$_G$ dal punto di vista dell'utente. Inoltre, consentono di identificare e comprendere rapidamente le relazioni e le interazioni tra i vari casi d'uso, offrendo una visione d'insieme delle funzionalità offerte dal *sistema*$_G$.

I diagrammi dei casi d'uso si concentrano sulla descrizione delle funzionalità del *sistema*$_G$ dal punto di vista degli utenti senza approfondire dettagli implementativi. La loro finalità principale è quella di evidenziare le interazioni dall'esterno al *sistema*$_G$, offrendo una visione focalizzata sulle funzionalità e sull'interazione dell'utente con il *sistema*$_G$ stesso.

Un diagramma dei casi d'uso fornisce una panoramica visuale delle interazioni chiave tra gli attori e il *sistema*$_G$, facilitando la comprensione dei requisiti funzionali del *sistema*$_G$ e la comunicazione tra gli *stakeholder*$_G$ del progetto.
Di seguito sono elencati i principali componenti di un diagramma dei casi d'uso:

- **Attori**

  Gli attori sono le entità esterne al *sistema*$_G$ che interagiscono con esso e possono essere utenti umani, altri sistemi *software*$_G$ o componenti esterne.
  Gli attori sono rappresentati come "stickman" all'esterno del rettangolo che delinea il *sistema*$_G$.

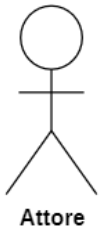

Figure 1: Rappresentazione Attore

- **Casi d'Uso**

  I casi d'uso identificano le diverse funzionalità offerte dal *sistema*$_G$ con cui l'attore può interagire.
  Ogni caso d'uso viene rappresentato tramite una forma ovale contenente un *ID*$_G$ ed un titolo esplicativo.

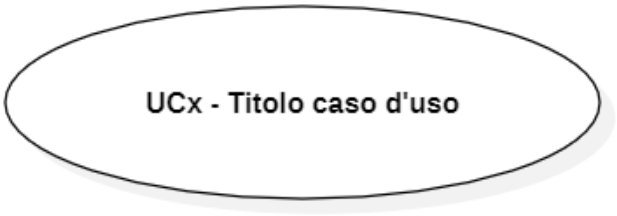

Figure 2: Rappresentazione caso d'uso

- **Sottocasi d'uso**

  Un sottocaso d'uso rappresenta una versione più dettagliata di un caso d'uso più generico, offrendo un livello di dettaglio più approfondito sulle funzionalità o sui particolari scenari di utilizzo rispetto al caso d'uso principale.

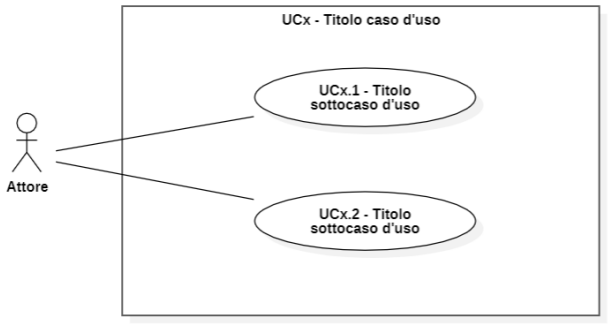

Figure 3: Rappresentazione sottocaso d'uso

- **Sistema**

  Il *sistema*$_G$ viene rappresentato da un rettangolo e viene identificato con un titolo. All'interno del *sistema*$_G$ saranno collocati i casi d'uso, mentre al suo esterno gli attori.

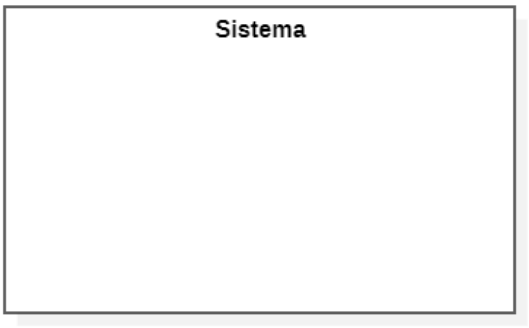

Figure 4: Rappresentazione sistema

- **Relazioni tra Attori e Casi d'Uso**

  - **Associazione**
    Le linee di associazione collegano gli attori ai casi d'uso corrispondenti, indicando quali attori sono coinvolti in una particolare interazione. Più precisamente, una linea di associazione collega un attore a un caso d'uso quando quell'attore è coinvolto nell'interazione descritta dal caso d'uso stesso. Questo legame rappresenta visivamente il ruolo dell'attore nell'utilizzo o nell'avvio di una funzione specifica offerta dal *sistema*$_G$.

Figure 5: Rappresentazione associazione

- **Relazioni tra attori**

  - **Generalizzazione tra attori**
    La generalizzazione tra attori rappresenta una relazione di ereditarietà, dove un attore specializzato (figlio) eredita comportamenti e caratteristiche da un attore base (genitore).
    Questo aiuta a organizzare gerarchicamente gli attori coinvolti nell'interazione con il *sistema*$_G$ nei diagrammi dei casi d'uso.
    Viene rappresentata con una linea solida e una freccia vuota che parte dall'attore figlio e arriva all'attore padre.

Figure 6: Rappresentazione generalizzazione tra attori

- **Relazioni tra casi d'uso**

  - **Inclusione**

    La relazione di inclusione indica che un caso d'uso (detto "includente") include l'esecuzione di un altro caso d'uso (detto "incluso").

    In pratica, quando un attore interagisce con il caso d'uso includente, il caso d'uso incluso viene eseguito come parte integrante del primo. Questo è utile per favorire il riutilizzo e per evitare duplicazione in diversi casi d'uso.

    La relazione di inclusione viene rappresentata da una freccia tratteggiata che collega il caso d'uso incluso al caso d'uso che lo include.

    Esempio: Supponiamo che per un applicazione e-commerce ci sia un caso d'uso "Conferma ordine". Questo caso d'uso può includere il caso d'uso "Visualizza carrello". Dopo la conferma dell'ordine, l'utente viene automaticamente reindirizzato alla visualizzazione del carrello al fine di ottenere una panoramica completa dei contenuti dell'ordine.

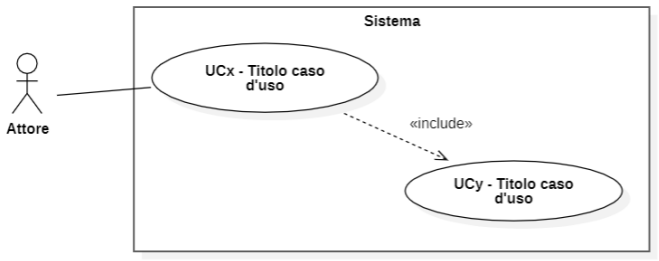

Figure 7: Rappresentazione inclusione

- **Estensione**
  La relazione di estensione indica che un caso d'uso (detto "estendente") può estendere il comportamento di un altro caso d'uso (detto "esteso") in determinate circostanze. In altre parole, l'esecuzione del caso d'uso estendente può essere estesa o arricchita dal caso d'uso esteso al verificarsi di determinate condizioni.
  La relazione di estensione è rappresentata da una freccia tratteggiata che collega il caso d'uso estendente al caso d'uso esteso.
  Esempio: Considera un caso d'uso "Visualizzazione errore di autenticazione". Questo caso d'uso potrebbe estendere il caso d'uso "Autenticazione" se durante la fase di autenticazione si inseriscono username e/o password non validi. In questo modo, l'estensione permette di gestire situazioni alternative senza ingombrare lo scenario principale di un caso d'uso e permettendo di evitare duplicazione.

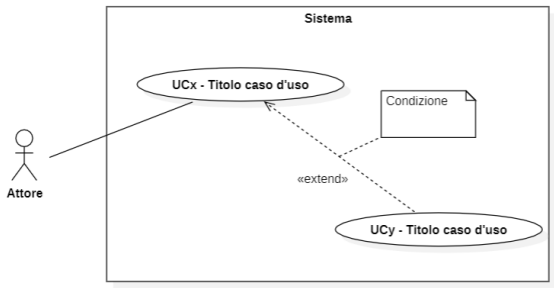

Figure 8: Rappresentazione estensione

– **Generalizzazione casi d'uso**
  La generalizzazione nei diagrammi dei casi d'uso rappresenta una relazione di ereditarietà tra casi d'uso, indicando che un caso d'uso più specifico eredita il comportamento da un caso d'uso più generico.
  Questa relazione è simboleggiata da una linea con una freccia vuota che punta dal caso d'uso più specifico al caso d'uso più generico.

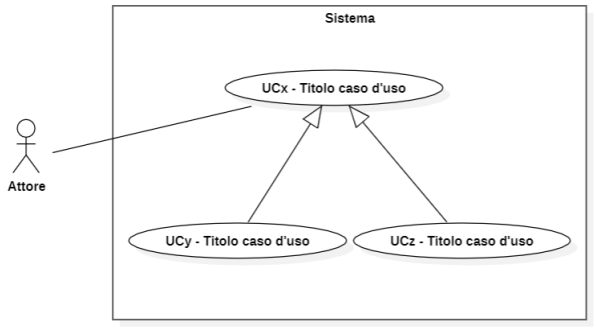

Figure 9: Rappresentazione generalizzazione

2.2.2.6 Requisiti

I requisiti di un prodotto *software*$_G$ sono specifiche dettagliate e documentate che delineano le funzionalità, le prestazioni, i vincoli e altri aspetti critici che il *software*$_G$ deve soddisfare. Questi requisiti fungono da guida per lo sviluppo, il testing e la valutazione del prodotto, assicurando che risponda alle esigenze degli utenti e agli obiettivi del progetto.
Includono:

- **Requisiti funzionali:** descrivono le funzionalità che il *software*$_G$ deve avere.
- **Requisiti non funzionali:** definiscono principalmente criteri di prestazione, qualità, sicurezza e vincoli del *sistema*$_G$, ovvero caratteristiche che non riguardano direttamente le funzionalità specifiche del *software*$_G$.

Una precisa definizione dei requisiti è fondamentale: devono risultare inequivocabili e rispondere pienamente alle attese del cliente o del *proponente*$_G$.

Ogni requisito è costituito da:

1. **Identificativo** nel formato:

   R [Abbreviazione tipologia requisito] [Codice]

   con:

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

2. **Importanza:**

   - **Obbligatorio:** irrinunciabile per qualcuno degli *stakeholder*$_G$;
   - **Desiderabile:** non strettamente necessario ma a valore aggiunto;
   - **Opzionale:** relativamente utile o contrattabile più avanti nel tempo.

3. **Descrizione:** narrazione chiara e dettagliata che fornisce una spiegazione completa del comportamento o della caratteristica che il *software*$_G$ deve possedere;

4. **Fonte:** fonte del requisito (ex. Capitolato, Verbale interno/esterno);

5. **Casi d'uso:** elenco casi d'uso associati.

2.2.2.7 Metriche

Le metriche nell'*analisi dei requisiti*$_G$ sono strumenti utilizzati per valutare, misurare e gestire diversi aspetti dei requisiti di un *sistema*$_G$ o di un progetto. Queste metriche aiutano a garantire che i requisiti siano completi, corretti, coerenti e comprensibili.

Metrica	Nome	Riferimento
M18PROS	Percentuale di Requisiti Obbligatori Soddisfatti	M18PROS
M19PRDS	Percentuale di Requisiti Desiderabili Soddisfatti	M19PRDS
M20PRPS	Percentuale di Requisiti Opzionali	M20PRPS

Table 2: Metriche relative all'attività di analisi dei requisiti

2.2.2.8 Strumenti

- **StarUML:** applicazione *software*$_G$ utilizzata dal team per la realizzazione dei diagrammi dei casi d'uso.

2.2.3 Progettazione

2.2.3.1 Descrizione

Lo scopo primario dell'*attività*$_G$ di progettazione consiste nell'identificare una soluzione realizzativa ottimale che soddisfi appieno le esigenze di tutti gli *stakeholder*$_G$, considerando i requisiti e le risorse disponibili.

La progettazione risponde alla domanda chiave: 'Qual è il modo migliore per realizzare ciò di cui c'è bisogno?'.

È cruciale definire l'*architettura*$_G$ del prodotto prima di avviare la fase di codifica, perseguendo correttezza per costruzione piuttosto che correttezza per correzione. Questo approccio permette di gestire in modo efficiente la complessità del prodotto, garantendo una struttura robusta e coesa durante l'intero processo di sviluppo.

2.2.3.2 Obiettivi

L'obiettivo principale è quindi quello di assicurare il soddisfacimento dei requisiti attraverso un *sistema*$_G$ di qualità delineato dall'*architettura*$_G$ del prodotto.
Questo comporta:

- Identificare parti componibili conformi ai requisiti, dotate di specifiche chiare e coese, sviluppandole con risorse sostenibili e costi contenuti;
- Organizzare il *sistema*$_G$ in modo da agevolare adattamenti futuri;
- Gestire la complessità del *sistema*$_G$ attraverso una progettazione dettagliata, suddividendo il *sistema*$_G$ in unità architetturali per semplificare la codifica di ciascuna parte, rendendola facilmente gestibile, veloce e verificabile.

Inizialmente, il team di progettazione condurrà un'analisi approfondita per selezionare con attenzione le tecnologie più adatte, valutandone attentamente i punti di forza, le debolezze ed eventuali criticità.

Una volta individuate le tecnologie appropriate, si procede allo sviluppo di un'*architettura*$_G$ di alto livello per comprendere e delineare la struttura generale del prodotto, costituendo così una base di partenza per la realizzazione del *PoC*$_G$. Questa *architettura*$_G$ fornisce una visione panoramica del *sistema*$_G$, identificando i principali componenti, i flussi di dati e le interazioni

tra di essi. Si presterà particolare attenzione a rendere il *sistema*$_G$ flessibile e adatto a potenziali modifiche future.

Successivamente, si avvierà lo sviluppo di un Proof of Concept (*PoC*$_G$), parte fondamentale della Technology Baseline, per valutare le decisioni prese riguardo all'*architettura*$_G$ e alle tecnologie adottate, nonché per verificarne l'aderenza agli obiettivi e alle specifiche del progetto.

In seguito allo sviluppo e ad un'attenta analisi del *POC*$_G$, si procederà con iterazioni aggiuntive, apportando miglioramenti, aggiustamenti e aggiunte, fino a giungere a un design completo. Questo design sarà fondamentale per lo sviluppo dell'*MVP*$_G$ (Minimum Viable Product), che costituirà una versione essenziale e funzionale del prodotto e che sarà parte della Product Baseline.

2.2.3.3 Documentazione

Specifica tecnica
Il documento descrive in modo approfondito il design definitivo del prodotto e fornisce istruzioni precise agli sviluppatori, guidandoli nella corretta implementazione della soluzione *software*$_G$ secondo i requisiti e le specifiche indicate. Ciò permette di ridurre la complessità e le ambiguità nel processo di sviluppo del *software*$_G$, contribuendo a garantire che il prodotto finale sia allineato alle aspettative del cliente e funzioni in modo ottimale.
Questo documento comprende diversi elementi cruciali:

- **Tecnologie utilizzate**: specifica le tecnologie e le *librerie*$_G$ di terze parti integrate nel *sistema*$_G$;
- **Architettura logica**: definisce i componenti, i ruoli, le connessioni e le interazioni nel *sistema*$_G$;
- **Architettura di deployment**: indica come le componenti architetturali vengono allocate e distribuite nel *sistema*$_G$ in esecuzione;
- **Pattern architetturali e design pattern**: descrive i design *pattern*$_G$ architetturali adottati e quelli influenzati dalle tecnologie utilizzate;
- **Vincoli e linee guida**: include restrizioni e regole da seguire durante lo sviluppo;
- **Procedure di testing e validazione**: indica i *processi*$_G$ per testare e verificare che il *software*$_G$ soddisfi i requisiti specificati;
- **Requisiti tecnici**: specifica in dettaglio i requisiti prestazionali, di sicurezza, di scalabilità e di compatibilità con determinate piattaforme che il *software*$_G$ deve soddisfare.

2.2.3.4 Qualità dell'architettura

- **Sufficienza**: l'*architettura*$_G$ soddisfa i requisiti funzionali e non funzionali definiti per il *software*$_G$, senza sovradimensionamento o sottodimensionamento;
- **Comprensibilità**: la chiarezza e la facilità con cui è possibile comprendere l'*architettura*$_G$ *software*$_G$, rendendo agevole per gli sviluppatori e gli *stakeholder*$_G$ capire come funziona il *sistema*$_G$;
- **Modularità**: l'*architettura*$_G$ è suddivisa in moduli o componenti chiaramente definiti, consentendo la separazione delle responsabilità e facilitando la manutenzione, l'aggiornamento e lo sviluppo parallelo;
- **Robustezza**: la capacità del *software*$_G$ di gestire situazioni anomale o errate senza interruzioni gravi o perdite di dati, mantenendo un funzionamento accettabile;
- **Flessibilità**: la facilità con cui il *sistema*$_G$ può essere adattato o esteso per soddisfare nuovi requisiti o cambiamenti senza richiedere modifiche radicali o strutturali;
- **Riusabilità**: la capacità di riutilizzare parti del *software*$_G$ in contesti diversi, riducendo lo sforzo di sviluppo e migliorando l'efficienza;
- **Efficienza**: il *software*$_G$ utilizza in modo ottimale le risorse disponibili, come memoria e CPU, per eseguire le sue funzioni nel minor tempo possibile;
- **Affidabilità**: la capacità del *software*$_G$ di svolgere correttamente le sue funzioni in modo coerente e prevedibile nel tempo;
- **Disponibilità**: il *software*$_G$ è accessibile e operativo quando richiesto, riducendo al minimo i tempi di inattività non pianificati;
- **Sicurezza rispetto a malfunzionamenti (Safety)**: la prevenzione di danni fisici o danni a persone o beni a causa di malfunzionamenti del *software*$_G$;
- **Sicurezza rispetto a intrusioni (Security)**: la protezione del *software*$_G$ da accessi non autorizzati, manipolazioni e intrusioni esterne;
- **Semplicità**: la capacità di mantenere l'*architettura*$_G$ *software*$_G$ il più semplice possibile senza compromettere la funzionalità o l'efficacia;
- **Incapsulazione**: il nascondere dei dettagli implementativi all'esterno di un componente, consentendo l'accesso solo attraverso un'interfaccia definita;
- **Coesione**: la misura in cui i componenti all'interno di un modulo o un'unità funzionano insieme per un obiettivo comune senza essere eccessivamente interdipendenti;

- **Basso accoppiamento**: il grado di dipendenza tra i diversi moduli o componenti del *software*$_G$, cercando di minimizzare le interazioni o le dipendenze tra di essi per migliorare la manutenibilità e la flessibilità del *sistema*$_G$.

2.2.3.5 Diagrammi UML

Vantaggi:

- **Chiarezza nella comunicazione**: rendono più comprensibili e chiari i concetti tecnici ai team e agli *stakeholder*$_G$;
- **Standardizzazione**: offrono un linguaggio comune per la documentazione e la comprensione dei sistemi *software*$_G$;
- **Analisi e progettazione visiva**: aiutano a identificare errori e lacune nel progetto prima dell'implementazione;
- **Modellazione e simulazione**: consentono la creazione di modelli predittivi, riducendo rischi e costi durante lo sviluppo;
- **Facilitano la manutenzione**: semplificano la comprensione della struttura del *software*$_G$, agevolando le operazioni di manutenzione;
- **Riducono errori di progettazione**: aiutano a individuare problemi prima dell'implementazione, riducendo correzioni costose;
- **Supportano la documentazione**: offrono una guida visiva per comprendere il *sistema*$_G$, inclusa nella documentazione del progetto.

A supporto dell'*attività*$_G$ di progettazione verranno utlizzati i **Diagrammi delle classi**.

Diagrammi delle classi

Ciascun diagramma delle classi rappresenta le proprietà e le relazioni tra le varie componenti di un *sistema*$_G$, offrendo una prospettiva statica e non a run time.

Le classi sono rappresentate graficamente sotto forma di rettangoli divisi in tre sezioni:

1. **Nome della classe**: identificativo della classe.
   È rappresentato in grassetto seguendo la convenzione PascalCase e deve esplicitare chiaramente la responsabilità della classe. Se la classe è astratta, il nome viene scritto in corsivo;

2. **Attributi**: ogni elemento sarà presentato in una riga separata, secondo il seguente formato:

   ```
   Visibilità Nome:  Tipo [Molteplicità] = Valore/i di Default
   ```

   - **Visibilità**: precede obbligatoriamente ogni attributo e rappresenta uno dei seguenti indicatori:
     - $-$: visibilità privata;
     - $+$: visibilità pubblica;
     - $\#$: visibilità protetta;
     - $\sim$: visibilità di package.
   - **Nome**: rappresenta univocamente l'attributo. Dev'essere rappresentativo dell'attributo e deve seguire la notazione `nomeAttributo:  tipo`.
     Qualora l'attributo fosse di tipo costante, allora il nome verrà scritto interamente in maiuscolo `NOMEATTRIBUTO: tipo`;
   - **Molteplicità**: nel caso di una sequenza di elementi, come liste o array, la sua lunghezza può essere specificata con la sintassi `tipoAttributo[molteplicità]`.
     Se la sequenza contiene un numero di elementi non conosciuto a priori, verrà adottata la sintassi `tipoAttributo[*]`.
     Nel caso di un singolo elemento, la sua dichiarazione è opzionale;
   - **Default**: ogni attributo può essere dichiarato con un valore di default.

3. **Firme dei metodi**: descrivono il comportamento delle classi individuate. Ogni metodo occuperà una riga e seguiranno il formato:

   ```
   Visibilità Nome (Parametri Formali):  Tipo di Ritorno
   ```

   - **Visibilità**: segue la convenzione definita per gli attributi;
   - **Nome**: rappresenta l'identificativo univoco del metodo e descrive in modo esplicito il suo obiettivo, seguendo la notazione *PascalCase*;
   - **Parametri formali**: il loro numero varia da 0 a n e sono separati tramite la virgola. Ogni parametro seguirà la notazione e le convenzioni definite per gli attributi;
   - **Tipo di ritorno**: determina il tipo che verrà ritornato dal metodo.

Convenzioni sui metodi

- I metodi getter, setter e i costruttori non vengono inclusi fra i metodi;
- I metodi astratti sono scritti in corsivo;
- I metodi statici sono sottolineati;
- L'assenza di attributi o metodi in una classe, determina una visualizzazione di campi vuoti nel diagramma delle classi.

Relazioni tra le classi

I diagrammi delle classi sono collegati tra loro mediante frecce che delineano le rispettive relazioni di dipendenza.
Qui di seguito, saranno delineate le rappresentazioni mediante frecce per le diverse relazioni:

- **Dipendenza**
  Se la classe A utilizza servizi, metodi o membri forniti dalla classe B, si dice che la classe A dipende dalla classe B e tale relazione viene rappresentata con una freccia tratteggiata dalla classe A alla classe B.
  La relazione di dipendenza indica il minor grado di accoppiamento tra due classi: un cambiamento nella classe B può influenzare la classe A, ma non viceversa.

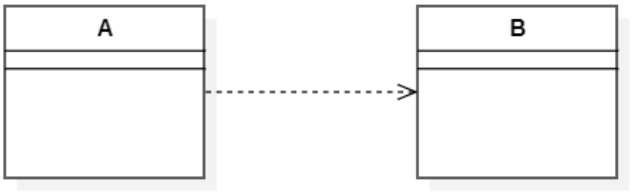

Figure 10: Diagramma delle classi di una relazione di Dipendenza.

- **Associazione**

  L'associazione tra due classi viene rappresentata tramite una linea continua e orientata. Questa connessione indica che la classe A contiene campi o istanze della classe B.
  Le molteplicità di occorrenza possono essere espresse tramite valori posizionati agli estremi della freccia:

  - $\mathbf{0}...\mathbf{1}$: A può possedere 0 o 1 istanza di B;
  - $\mathbf{0}...*$: A può possedere 0 o più istanze di B;

- $\mathbf{1}$: A possiede esattamente un'istanza di B (in questo caso non è necessario specificare la molteplicità);
- $*$: A possiede più istanze di B;
- $\mathbf{n}$: A possiede esattamente n istanze di B.

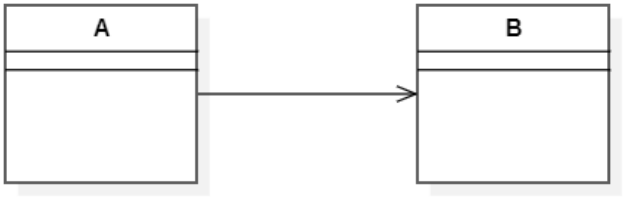

Figure 11: Diagramma delle classi di una relazione di Associazione.

- **Aggregazione**

  Rappresentata con una freccia a diamante vuota.

  Indica una relazione "parte di" (part of) in cui una classe è composta da diverse parti (altre classi), ma le parti possono esistere anche indipendentemente dalla classe principale. Nella figura la classe B è parte della classa A.

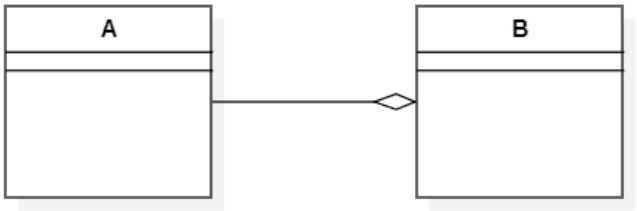

Figure 12: Diagramma delle classi di una relazione di Aggregazione.

- **Composizione**

  Rappresentata con una freccia a diamante piena. È simile all'aggregazione, ma le parti (altre classi) sono strettamente dipendenti dalla classe principale e non possono esistere indipendentemente.
  Nella figura la classe B è parte della classe A ed esiste solo come parte di A.

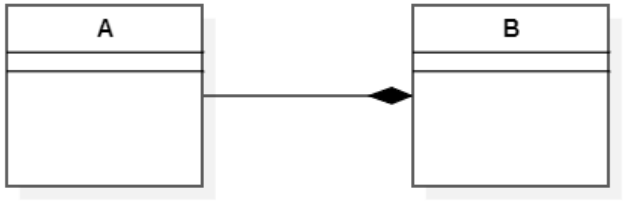

Figure 13: Diagramma delle classi di una relazione di Composizione.

- **Generalizzazione**

  Rappresentata con una freccia continua vuota. Rappresenta la relazione "is a" (è un) tra una classe genitore (superclasse) e una classe figlia (sottoclasse). La classe figlia eredita attributi e comportamenti dalla classe genitore.
  Equivale all'ereditarietà nei linguaggi di programmazione.
  Le proprietà della superclasse non si riportano nel diagramma della sottoclasse, a meno di override.

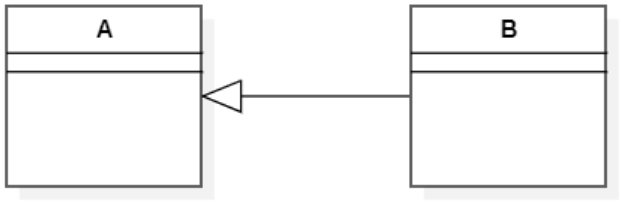

Figure 14: Diagramma delle classi di una relazione di Generalizzazione.

- **Interface Realization**

  La relazione di "interface realization" indica che una classe fornisce l'implementazione dei metodi definiti in un'interfaccia specifica. Questa relazione è importante quando si desidera mostrare come una classe concreta soddisfi i requisiti di un'interfaccia specifica definendo e implementando i suoi metodi.

  Nella figura l'interfaccia A, rappresentata tramite un cerchio, viene implementata dalla classe B e questa relazione viene rappresentata graficamente con una linea da B ad A.

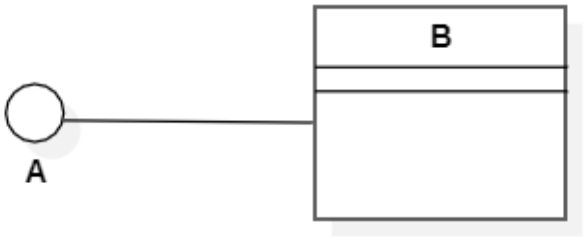

Figure 15: Diagramma delle classi di una relazione di Interface Realization.

2.2.3.6 Design Pattern

I design *pattern*$_G$ costituiscono una risposta consolidata a problemi di progettazione che si presentano ciclicamente in determinati contesti. Questi modelli offrono un approccio di progettazione riusabile, garantendo qualità nella soluzione ed una rapida implementazione. L'adozione di un design *pattern*$_G$ avviene soprattutto quando una soluzione si è dimostrata efficace in un contesto specifico.

Solitamente vengono fornite guide dettagliate sull'applicazione dei *pattern*$_G$, delineando il loro utilizzo ottimale. Ogni design *pattern*$_G$ deve essere accompagnato da una rappresentazione grafica illustrativa del suo funzionamento, da una spiegazione testuale della sua logica e da una descrizione della sua utilità all'interno dell'*architettura*$_G$ complessiva.

Questa documentazione svolge un ruolo chiave nel favorire una comprensione approfondita dell'*integrazione*$_G$ del design *pattern*$_G$ nell'*architettura*$_G$ generale e nella prevenzione di errori di progettazione.

2.2.3.7 Test

Nell'ambito del processo di sviluppo, il testing riveste un ruolo fondamentale nell'assicurare la qualità del prodotto finale. Durante questa fase, si delineano i requisiti di testing, si definiscono i casi di *test*$_G$ e i criteri di accettazione, fungendo da strumenti per valutare il *software*$_G$.

Il suo obiettivo principale è individuare e risolvere eventuali problemi o errori presenti nel *software*$_G$ prima del rilascio del prodotto finale. In aggiunta, il processo di testing è essenziale per garantire che il *software*$_G$ soddisfi le specifiche e le aspettative del cliente.

I progettisti avranno il completo controllo di questa *attività*$_G$, compresa la definizione dei *test*$_G$ da eseguire. Inoltre, nella sezione 3.2.4, si può trovare una descrizione dettagliata delle varie tipologie di *test*$_G$ e della terminologia da adottare, offrendo così un'ulteriore chiarezza su questa fase critica del processo di sviluppo del *software*$_G$.

2.2.3.8 Metriche

Metrica	Nome	Riferimento
M25ATC	Accoppiamento tra classi (ATC)	M25ATC
M30PG	Profondità delle Gerarchie (PG)	M30PG
M31TMR	Tempo Medio di Risposta (TMR)	M31TMR
M32FU	Facilità di Utilizzo (FU)	M32FU
M33TA	Tempo di Apprendimento (TA)	M33TA
M34VBS	Versioni dei Browser Supportate (VBS)	M34VBS

Table 3: Metriche relative all'attività di progettazione

2.2.3.9 Strumenti

- **StarUML:** applicazione *software*$_G$ utilizzata per la realizzazione dei diagrammi delle classi.

2.2.4 Codifica

2.2.4.1 Descrizione

L'*attività*$_G$ di codifica è affidata al ruolo del programmatore e rappresenta il momento cruciale in cui le funzionalità richieste dal *proponente*$_G$ prendono forma.
Durante questa fase, le idee e i concetti delineati dai progettisti vengono tradotti in codice, creando istruzioni e procedure che i calcolatori possono eseguire.

I programmatori devono rispettare scrupolosamente le linee guida e le norme stabilite per garantire che il codice sia in linea con le specifiche stabilite e che traduca in modo accurato le concezioni iniziali dei progettisti.

2.2.4.2 Obiettivi

La codifica è finalizzata alla creazione di un prodotto *software*$_G$ in linea con le richieste del *committente*$_G$ e conforme agli accordi stipulati.
Il rigoroso rispetto delle norme garantisce la creazione di codice di alta qualità, facilitando la manutenzione, l'estensione e la verifica del *software*$_G$, contribuendo costantemente al miglioramento della sua qualità complessiva.

2.2.4.3 Norme di codifica

Le seguenti norme sono state formalizzate prendendo spunto dal libro "Clean Code" di Robert C. Martin.

- **Nomi significativi**: usare nomi che riflettano il significato e lo scopo delle variabili, funzioni e classi. Evitare abbreviazioni ambigue;
- **Indentazioni e formattazione consistente**: l'utilizzo di un tab per ciascun livello di annidamento è fondamentale per assicurare una struttura coerente del codice, migliorandone la comprensione e agevolandone la gestione;
- **Lunghezza dei metodi**: la lunghezza ottimale dei metodi può variare a seconda del contesto e delle best practices di programmazione. Tuttavia, in generale, molti esperti consigliano che i metodi siano brevi e focalizzati su una singola responsabilità. Un principio comune è quello espresso da Robert C. Martin nel suo libro "Clean Code", che suggerisce che i metodi dovrebbero essere idealmente lunghi quanto basta per svolgere una singola operazione e non più lungo di quanto si possa visualizzare senza dover scorrere la pagina.

  Questo favorisce:

  - **Chiarezza**;
  - **Manutenibilità**;
  - **Comprensibilità**;
  - **Testabilità:** metodi più brevi sono più facili da testare in isolamento, il che favorisce l'implementazione di *test*$_G$ di unità efficaci;
  - **Conformità ai principi SOLID:** la brevità dei metodi è spesso correlata al principio Single Responsibility Principle (*SRP*$_G$) dei principi *SOLID*$_G$, il che favorisce la costruzione di codice più modulare e coeso.

- **Lunghezza righe di codice**: mantenere le righe di codice entro i 80-120 caratteri permette una migliore leggibilità del codice su schermi di diverse dimensioni e facilita la visualizzazione di più file affiancati. Inoltre, limitare la lunghezza delle righe può incoraggiare la scrittura di codice più chiaro e modulare;
- **Commenti**
  Evitare commenti superflui e non necessari: l'obiettivo è scrivere il codice in modo chiaro e autoesplicativo, riducendo la dipendenza da commenti esplicativi.

2.2.4.4 Strumenti

- **Visual Studio Code:** IDE utilizzato dal team di sviluppo per la codifica del prodotto *software*$_G$.

2.2.4.5 Metriche

Metrica	Nome	Riferimento
M15SC	Statement Coverage (MSC_G)	M15SC
M16BC	Branch Coverage (MBC)	M16BC
M17CNC	CoNdition Coverage (CNC)	M17CNC
M26MCCM	Complessità Ciclomatica per Metodo ($MCCM_G$)	M26MCCM
M27PM	Parametri per metodo (PM)	M27PM
M28APC	Attributi per classe (APC)	M28APC
M29LCM	Linee di codice per metodo (LCM)	M29LCM

Table 4: Metriche relative all'attività di codifica

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

La scrittura dei file *Docker*$_G$ è considerata parte del processo di sviluppo del *software*$_G$.
Le regole e le *best practice*$_G$ di codifica per i file *Docker*$_G$ sono fondamentali per garantire la creazione, la gestione e la distribuzione efficace dei *container*$_G$:

- **Chiarezza e Coerenza:**
  - Utilizzare nomi descrittivi per le immagini e i *container*$_G$.
  - Mantenere una struttura coerente e consistente per assicurare un'organizzazione uniforme all'interno dei file *Docker*$_G$.
- **Versionamento:**
  - Specificare sempre la versione dell'immagine di base (base image) per garantire la riproducibilità.
  - Evitare di utilizzare tag "latest" per immagini di produzione.
- **Minimizzazione degli strati (Layering):**
  - Ridurre il numero di istruzioni nell'esecuzione del *Dockerfile*$_G$ per minimizzare gli strati dell'immagine.
  - Raggruppare le istruzioni correlate per sfruttare la cache *Docker*$_G$.
- **Sicurezza:**
  - Utilizzare immagini ufficiali o verificate.
  - Evitare di eseguire *processi*$_G$ *Docker*$_G$ con privilegi elevati quando possibile.
  - Usare ARG per parametrizzare informazioni sensibili.
- **Ottimizzazione delle risorse:**
  - Limitare l'uso di risorse all'interno dei *container*$_G$ (CPU, memoria, etc.).
  - Usare immagini leggere e ottimizzate per la produzione.
- **Gestione delle variabili d'ambiente:**
  - Usare variabili d'ambiente per configurazioni dinamiche.
  - Fornire valori predefiniti opportuni per le variabili d'ambiente.

- **Logging e monitoraggio:**
  - Configurare i *container*$_G$ per registrare i *log*$_G$ in modo efficace.
  - Integrare strumenti di monitoraggio, se necessario.
- **Pulizia e riduzione delle dimensioni:**
  - Pulire i pacchetti temporanei e le risorse non necessarie dopo l'installazione delle dipendenze.
  - Ridurre le dimensioni delle immagini utilizzando *multi-stage builds*$_G$.
- **Documentazione:**
  - Aggiungere commenti nel *Dockerfile*$_G$ per spiegare le decisioni di progettazione.
  - Fornire una documentazione chiara su come utilizzare e configurare il *container*$_G$.
- **Testing:**
  - Implementare *test*$_G$ automatizzati per il *Dockerfile*$_G$ e i *container*$_G$, se possibile.

2.2.5.2 Strumenti

- **Visual Studio Code:** IDE utilizzato dal team di sviluppo per la codifica dei file *Docker*$_G$.
- **Docker:** *piattaforma*$_G$ di virtualizzazione leggera che permette di creare e gestire applicazioni in contenitori *software*$_G$, fornendo un ambiente portatile, scalabile ed efficiente per lo sviluppo, il testing e la distribuzione delle applicazioni.

3 Processi di supporto

3.1 Documentazione

3.1.1 Introduzione

La documentazione costituisce l'insieme di informazioni scritte che accompagnano un prodotto *software*$_G$, fornendo dettagli utili a sviluppatori, distributori e utenti.
Tra gli scopi della documentazione troviamo:

- Comprensione del prodotto senza supporto umano, usandola come mezzo di comunicazione;
- Segnare il confine tra creatività e disciplina;

- Assicurare che i *processi*$_G$ produttivi si svolgano con la qualità attesa.

L'obiettivo della sezione è:

- Definire delle procedure ripetibili che permettano di uniformare la documentazione prodotta dal gruppo ed il metodo di lavoro;
- Raccogliere ed organizzare le norme che i membri del team devono seguire così da semplificare l'operazione di scrittura dei documenti.

Tali norme dovranno essere applicate da tutti i membri del team, ed i sorgenti LaTeX che contengono tale documentazione verranno inseriti nel *repository*$_G$ Sorgente-documenti.

3.1.2 Documentation as Code

L'approccio che si intende adottare è quello di "Documentation as Code" (Documentazione come Codice) che consiste nel trattare la documentazione di un progetto *software*$_G$ allo stesso modo in cui si tratta il codice sorgente.
Questo approccio è incentrato sull'utilizzo di pratiche e strumenti tipici dello sviluppo *software*$_G$ per creare, gestire e distribuire la documentazione. Alcuni aspetti chiave della "Documentation as Code" includono:

- **Versionamento**;
- **Scrittura in formato testuale**;
- **Automazione**;
- **Collaborazione**;
- **Integrazione Continua**;
- **Distribuzione**.

Questo approccio porta diversi vantaggi, tra cui una maggiore coerenza, una migliore tracciabilità delle modifiche e facilità di manutenzione. Inoltre, il concetto di "Documentation as Code" si allinea con la filosofia DevOps, dove la collaborazione e l'automazione sono valori chiave.

3.1.3 Sorgente documenti

Poiché i documenti vengono redatti in LaTeX, per favorire una migliore collaborazione tra diversi autori, si è scelto di creare un file sorgente per ciascuna sezione e sottosezione di ogni documento. Questo approccio consente a ciascun autore di lavorare sulle singole sezioni o

sottosezioni, evitando di apportare modifiche a parti del documento non pertinenti al proprio ambito.
Infine, viene creato un file principale con il nome del documento, nel quale verranno collegati tutti i file delle varie sezioni e sottosezioni mediante l'uso del comando `\input{Sezione.tex}` o, nel caso delle sottosezioni, `\input{Sottosezione.tex}`.
Nel caso dei verbali, essendo documenti composti da sezioni molto brevi, non si utilizza questo approccio.

3.1.4 Ciclo di vita dei documenti

Il ciclo di vita dei documenti è una sequenza di stati e *attività*$_G$:

1. **Stato: Necessità**
   (a) Nasce la necessità di una documentazione, per obbligo o per opportunità;
   (b) Pianicazione della sua stesura;
   (c) Suddivisione in sezioni;
   (d) Durante le riunioni, si procede alla discussione e alla definizione collettiva di una traccia per il contenuto;
   (e) Assegnazione delle sezioni ai redattori tramite task su ITS.
2. **Stato: Redazione**
   (a) Ogni tipo di documento viene creato secondo la struttura specificata nella *sezione 3.1.6*;
   (b) Il team realizza il documento redigendone il contenuto rispettando le norme definite.
3. **Stato: Verifica**
   (a) Quando la redazione del documento è conclusa, questo viene revisionato dai verificatori;
   (b) Il documento viene compilato e il PDF generato viene inserito nella repository.
4. **Stato: Manutenzione**
   (a) Manutenzione: Ogni documento soggetto a *configuration management*$_G$ deve essere adeguatamente modificato in conformità alle regole di versionamento e *change management*$_G$;
   (b) La richiesta di modifica derivata da nuove esigenze relative al contenuto del documento determina il ritorno del documento allo stato di modifica.

3.1.5 Procedure correlate alla redazione di documenti

3.1.5.1 I redattori

Il redattore responsabile della redazione di un documento o di una sua sezione deve seguire lo stesso approccio richiesto per la codifica del *software*$_G$, adottando il *workflow*$_G$ noto come "Feature Branch".

Caso redazione nuovo documento/sezione o modifica documento/sezione già verificato/a

Dalla *repository*$_G$ Sorgente-documenti contenente i sorgenti LaTeX, il redattore dovrà creare un nuovo *branch*$_G$ Git in locale e spostarsi su di esso mediante l'utilizzo del comando:

```
git checkout main
git checkout -b identificativoBranch
```

Il nome del *branch*$_G$ destinato alla redazione o modifica del documento o di una sua sezione deve essere descrittivo al fine di consentire un'identificazione immediata e precisa del documento o della sezione su cui si sta lavorando. Pertanto, relativamente alla redazione dei documenti, devonono essere adottatate le specifiche ***convenzioni per la nomenclatura dei branch***.
A questo punto, dopo aver creato/modificato il documento o la sezione assegnata e avviato la stesura, affinché gli altri redattori possano continuare il lavoro, è necessario rendere il *branch*$_G$ accessibile anche nella *repository*$_G$ remota, seguendo i seguenti passaggi:

```
git add .
git commit -m "Descrizione del commit"
git push origin identificativoBranch
```

Caso modifica documento in stato di redazione

Qualora il redattore intenda continuare la stesura di un documento (o sezione) iniziato da un altro redattore, sono necessari i comandi:

```
git pull
git checkout identificativoBranch
```

Salvataggio e condivisione progressi di task non completate

Alla fine di una sessione di modifiche di un file, nel caso si desideri rendere accessibile ai membri il lavoro non ancora completato e, pertanto, non pronto alla verifica, è necessario assicurarsi di essere nel *branch*$_G$ corretto e successivamente:

1. Eseguire il push$_G$ delle modifiche fatte nel *branch*$_G$

```
git add .
git commit -m "Descrizione del commit"
git push origin identificativoBranch
```

2. Nel caso di problemi con il punto 1:

```
git pull origin identificativoBranch
```

3. Risolvi i conflitti e ripeti punto 1.

Completamento attività di redazione

Una volta terminata la redazione di un documento o di una sezione, ogni redattore deve:

1. Segnalare il completamento dell'*attività*$_G$ a loro assegnata (essa sia la stesura completa di un documento o di una sua parte) posizionando l'*issue*$_G$ relativa nella colonna "Da revisionare" della DashBoard documentazione.
2. Attuare una ***Pull Request:***
   (a) Aggiornare il registro delle modifiche inserendo i dati richiesti in una nuova riga e incrementando la versione (il verificatore è definito all'assegnazione della task, presente nella descrizione della Issue del'ITS). È importante sottolineare che l'incremento della versione deve avvenire seguendo la convenzione specificata nella *sezione 3.4.2* e tramite il comando "`{\label{Git_Action_Version}X.Y.Z}`", come specificato nel paragrafo "*Informazioni utili*" relativo alla sezione dedicata al registro delle modifiche;
   (b) Eseguire i passaggi dettagliati nel caso "***Salvataggio e condivisione progressi di task non completate***";
   (c) Accedere al repository GitHub » Apri la sezione "Pull requests" » Clicca "New Pull Request";
   (d) Selezionare come *branch*$_G$ di destinazione "main" e come *branch*$_G$ sorgente il ramo utilizzato per la redazione del documento (o sezione) da validare;
   (e) Cliccare "Create Pull Request";
   (f) Inserire un titolo significativo e una breve descrizione alla Pull Request » Seleziona i verificatori su "Reviewers" » Clicca "Create Pull Request".

Se i verificatori non convalidano il documento (o la sezione), i redattori riceveranno dei feedback allegati alla Pull Request relativi ai problemi identificati.

3.1.5.2 I Verificatori

I compiti e le procedure dei verificatori sono dettagliate al *paragrafo 3.2.2*.

3.1.5.3 Il responsabile

Nel processo di redazione dei documenti, il compito del responsabile consiste nel:

- **Identificare i documenti o le sezioni da redigere**;
- **Stabilire le scadenze entro cui devono essere completati**;
- **Assegnare i redattori e i verificatori ai task**;
- **Approvazione:** il responsabile si riserva il diritto di approvare il lavoro o richiedere eventuali ulteriori modifiche su tutti i documenti redatti e verificati nel periodo in cui ha esercitato la propria carica.

3.1.5.4 L'amministratore

Nel processo di redazione dei documenti, il compito dell'amministratore consiste nell'inserire nell'ITS le *attività*$_G$ specificate dal responsabile:

- Redattori: assegnatari della *issue*$_G$;
- Verificatori: specificati nella descrizione della *issue*$_G$;
- Scadenza.

3.1.6 Struttura del documento

I documenti seguono un rigoroso e uniforme schema strutturale, il quale richiede un'osservanza scrupolosa.

3.1.6.1 Registro delle modifiche

Tutti i documenti devono essere provvisti di un registro delle modifiche in formato tabellare che contiene un riassunto delle modifiche apportate al documento nel corso del tempo.
La tabella deve essere inserita nella sezione registro delle modifiche subito prima dell'indice del documento e deve registrare le seguenti informazioni:

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

Informazioni utili:

- La convezione per il versionamento è presente alla *sezione 3.4.2*;
- La prima riga del registro delle modifiche identifica la versione attuale del documento;
- Durante la compilazione del file sorgente LaTeX, viene eseguito un processo automatizzato che integra la versione corretta nel nome del documento, basandosi sulla prima riga del registro delle modifiche. Affinché ciò avvenga correttamente, è necessario specificare l'ultima versione all'interno del registro delle modifiche del documento utilizzando il comando `{\label{Git_Action_Version}X.Y.Z}`.

3.1.6.2 Prima pagina

Nella prima pagina di ogni documento deve essere presente:

- Nome e mail del gruppo;
- Nome del documento;
- Redattori;
- Verificatori;
- Destinatari.

3.1.6.3 Indice

Tutti i documenti devono essere provvisti di un indice dove saranno elencate le varie sezioni e sottosezioni, con la possibilità di raggiungerle direttamente tramite click. In caso di presenza di figure nel documento, sarà presente anche un indice relativo.

3.1.6.4 Pié di pagina

In ogni pié di pagina deve essere presente:

- Nome del gruppo;
- Nome del documento;
- Numero di pagina.

3.1.6.5 Verbali: struttura generale

I verbali assumono l'importante ruolo di costituire un registro ufficiale dei meeting, atto a riportare in maniera accurata gli argomenti trattati, le decisioni adottate, le azioni da intraprendere e le figure coinvolte.

In particolare, è possibile distinguere tra verbali interni, destinati all'uso interno dell'organizzazione, e verbali esterni, che trovano applicazione quando vi sono terze parti coinvolte nelle discussioni o nelle decisioni documentate.
La struttura dei verbali è la seguente:

- **Prima pagina**
  Oltre alle informazioni comuni a ogni documento vengono specificate:
  - Data della riunione e tipologia (Interna,Esterna) nel nome del documento;
  - Luogo (nel caso non sia un luogo fisico si specifica il canale di comunicazione adottato);
  - Ora di inizio e fine dell'incontro;
  - Amministratore;
  - Partecipanti della riunione (interni ed eventuali esterni);
  - Il Responsabile (in basso a destra);
- **Corpo del documento**
  Nel corpo del documento sono presenti le seguenti sottosezioni:
  - **Revisione del periodo precedente**
    Si analizza lo stato delle *attività*$_G$ e, relativamente all'approccio lavorativo, si valutano gli aspetti positivi e le difficoltà incontrate in modo tale da identificare azioni di miglioramento per ottimizzare i *processi*$_G$;
  - **Ordine del giorno**
    Elenco delle tematiche discusse durante la riunione, accompagnate dai relativi esiti. Nel caso del verbale esterno, se sono presenti richieste di chiarimenti effettuate alle terze parti coinvolte, saranno incluse nella sottosezione "**Richieste e chiarimenti**".
  - **Attività da svolgere** (solo nel caso dei verbali interni):
    Tabella dove ogni riga identifica un'*attività*$_G$ e in cui viene specificato:
    - Nome della task da svolgere;
    - Id *issue*$_G$ dell'ITS;
    - Verificatore dell'*attività*$_G$.
- **Ultima pagina**
  Nel caso di verbale esterno, in ultima pagina, deve essere presente la firma delle terze parti coinvolte, il luogo e la data.

Il template per i verbali è raggiungibile tramite il seguente link.

3.1.7 Regole tipografiche

Documenti del progetto e nome dei file

Il nome dei documenti generati deve essere omogeneo, con la prima lettera maiuscola ed il resto minuscolo e, ad eccezione dei verbali, deve contenere un riferimento alla versione del documento (vedi notazione per il versionamento nella *sezione 3.4.2*).
Nello specifico devono seguire la seguente convenzione:

- **Verbali:** Verbale_AAAA-MM-DD ;
- **Norme di Progetto:** 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:** Glossario_vX.Y.Z .

Regole sintattiche:

- I titoli delle sezioni iniziano con la lettera maiuscola;
- Ciascuna voce di un elenco deve essere seguita da un punto e virgola, ad eccezione dell'ultima che deve terminare con un punto;
- Le date vengono scritte nel formato GG/MM/AAAA (giorno/mese/anno);
- I numeri razionali si scrivono utilizzando la virgola come separatore tra parte intera e parte decimale;
- Quando si fa riferimento a informazioni contenute in un documento diverso da quello attuale, è fondamentale indicare la versione del documento e, se necessario, specificare anche la sezione pertinente (Ex: Norme di Progetto v.1.0.0 sez. 1.2.3).

Stile del testo:

- **Grassetto**:
  - Titoli delle sezioni;
  - Parole o frasi ritenute di rilievo.
- **Italico**:
  - Riferimenti a paragrafi, sezioni e documenti;
  - Nomi delle aziende;
  - Termini presenti nel *Glossario*.

Riferimenti

- Per i riferimenti interni si usa il segunte colore: RGB(0.0,0.0,0.8);
- Per i riferimenti esterni si usa il colore: RGB(0.0, 0.5, 0.0).

3.1.8 Abbreviazioni

Nei documenti vi sono molte ripetizioni di termini per la quale si possono usare le seguenti abbreviazioni:

Abbreviazione	Scrittura Estesa
RTB_G	Requirements and Technology Baseline
PB_G	Product Baseline
CA	Customer Acceptance
ITS	Issue Tracking System
CI	Configuration Item
SAL_G	Stato Avanzamento Lavori

3.1.9 Strumenti

Gli strumenti utilizzati dalle *attività*$_G$ di redazione dei documenti vogliono soddisfare il principio adottato di "Documentation as Code".

- **GitHub**: utilizzato per il versionamento, la collaborazione e la distribuzione dei documenti, nonché per automatizzare la loro compilazione;
- LaTeX: utilizzato per la redazione dei documenti, sfruttando il suo potente *sistema*$_G$ di formattazione testuale per creare documenti di alta qualità;
- **Visual Studio Code**: IDE utilizzato dai redattori per la redazione dei documenti grazie al supporto del plugin **LaTeX Workshop**.

3.2 Verifica

3.2.1 Introduzione

La verifica è un processo essenziale che accompagna l'intero ciclo di vita del *software*$_G$, partendo dalla fase di progettazione fino alla manutenzione.

Il suo obiettivo principale è garantire l'efficienza e la correttezza delle *attività*$_G$, istanziando per ciascun prodotto un processo di verifica mirato a garantire risultati conformi alle aspettative. Questo processo si avvale di tecniche di analisi e *test*$_G$ per assicurare che i prodotti soddisfino i requisiti specificati.

Per assicurare il corretto svolgimento della verifica, è fondamentale seguire alcune linee guida chiave, tra cui l'utilizzo di criteri affidabili e di procedure ben definite. Questo assicura che il prodotto si trovi in uno stato stabile, pronto per passare alla successiva fase di validazione.

La verifica viene eseguita su tutti i *processi*$_G$ in esecuzione, quando si raggiunge un adeguato livello di maturità o in seguito a modifiche dello stato del processo. Durante questa fase, si analizza la qualità dei prodotti e dei *processi*$_G$ utilizzati per garantire la conformità agli *standard*$_G$ di qualità definiti.

Le *attività*$_G$ di verifica sono assegnate a verificatori, i quali, seguendo l'ordine definito nel modello a V, analizzano i prodotti e valutano la loro conformità ai requisiti di qualità specificati nel *Piano di Qualifica*.

È fondamentale documentare nel *Piano di Qualifica* tutte le *attività*$_G$ che compongono il processo di verifica, descrivendone gli obiettivi, i risultati attesi e quelli ottenuti. Questo fornisce linee guida per una valutazione accurata della qualità, normando e rendendo ripetibile il processo di verifica.

3.2.2 Verifica dei documenti

Il ruolo del verificatore nei documenti è cruciale per garantire la qualità e l'accuratezza del contenuto.
Quando il verificatore individua un'Issue nella colonna "Da revisionare" della DashBoard documentazione, sarà tenuto a convalidare il file corrispondente presente nella *repository*$_G$ Sorgente documenti.
In aggiunta, il revisore riceverà una notifica via email quando il redattore completa la propria *attività*$_G$, comunicandogli la presenza della Pull Request assegnatagli.
Nella sezione "Pull requests" di GitHub, il revisore troverà la richiesta di unire il *branch*$_G$ di redazione al *branch*$_G$ "main". Accedendo alla Pull Request su GitHub, il revisore avrà la possibilità di esaminare attentamente il documento in questione.
Durante questa revisione, il revisore potrà aggiungere commenti direttamente sul documento, fornendo feedback e suggerimenti ai redattori per eventuali modifiche necessarie per la validazione. I commenti aggiunti saranno visibili ai redattori, consentendo loro di comprendere le richieste di revisione e apportare le modifiche richieste. Questo processo collaborativo facilita il flusso di lavoro di revisione e garantisce la qualità e

l'accuratezza del documento finale.

Per validare un documento le verifiche da attuare sono:

- **Revisione della correttezza tecnica:** eseguire una revisione tecnica del documento per garantire che tutte le informazioni siano corrette, coerenti e rispettino le norme stabilite;
- **Conformità alle norme:** verificare che il documento segua le linee guida e gli *standard*$_G$ prestabiliti per la formattazione, la struttura e lo stile;
- **Consistenza e coerenza:** assicurarsi che il documento sia consistente internamente e coerente con altri documenti correlati. Verificare che non ci siano discrepanze o contraddizioni;
- **Chiarezza e comprensibilità:** valutare la chiarezza del testo, assicurandosi che il linguaggio sia comprensibile per il pubblico di destinazione e che non ci siano ambiguità;
- **Revisione grammaticale e ortografica:** controllare la grammatica, l'ortografia e la punteggiatura del documento per garantire una presentazione professionale;

Dopo aver accertato il rispetto dei criteri sopra citati, qualora fossero soddisfatti, il procedimento per convalidare il documento è il seguente:

1. **Accetta la Pull Request:** accedere alla pagina della Pull Request in cui si agirà con il ruolo di revisore nel *repository*$_G$ Sorgente-documenti su GitHub » Risolvere eventuali conflitti » Merge Pull Request;
2. **Elimina il branch:** eliminare il *branch*$_G$ creato per la redazione del documento (o sezione);
3. **Sposta la issue in Done:** nella DashBoard documentazione spostare la *issue*$_G$ relativa al documento validato dalla sezione "Da revisionare" a "Done";
4. **Controllo generazione PDF:** un'automazione tramite GitHub Actions compilerà il file LaTeX e genererà automaticamente il PDF nel *branch*$_G$ main della *repository*$_G$ Documents con la data di redazione nel caso il documento sia un verbale, oppure la versione aggiornata per tutte le altre tipologie di file.
   Se la procedura di compilazione dovesse fallire, il verificatore riceverà automaticamente una notifica via mail e dovrà accedere alla sezione "Actions" di GitHub per esaminare l'origine dell'errore e intraprendere le azioni necessarie per correggerlo.

3.2.3 Analisi

Attività di controllo su oggetti statici (documentazione, codice) e dinamici (componenti *software*$_G$).

3.2.3.1 Analisi statica

Questo genere di analisi prende il nome di "statica" proprio perché viene condotta sul prodotto senza la necessità di eseguirlo.
L'efficacia di questa pratica dipende in modo diretto dall'esperienza e dalla competenza dei verificatori coinvolti. Essa coinvolge l'utilizzo di metodi di lettura, sia manuali che automatici, volti a individuare eventuali errori formali come violazione di vincoli, presenza di difetti e manifestazione di proprietà indesiderate nel prodotto in questione.
Le due principali tecniche impiegate sono:

3.2.3.2 Walkthrough

Il verificatore controlla nella totalità il prodotto in cerca di difetti o errori, senza svolgere una ricerca specifica per un certo tipo di errore. Questa metodologia implica appunto una verifica approfondita della documentazione e del codice attraverso un'analisi sistematica. Si promuove la collaborazione e l'interazione tra il verificatore e l'autore del prodotto, strutturata nei seguenti quattro passaggi:

1. **Pianificazione:** una fase di dialogo tra gli autori e i verificatori, finalizzata all'identificazione delle proprietà e dei vincoli che il prodotto deve soddisfare per essere considerato corretto;
2. **Lettura:** il verificatore esamina attentamente i documenti e/o il codice, individuando errori e verificando la conformità ai vincoli prestabiliti;
3. **Discussione:** successivo confronto tra autori e verificatori per discutere l'esito della lettura e valutare eventuali correzioni necessarie;
4. **Correzione:** *attività*$_G$ assegnata agli autori per correggere gli errori individuati nei passaggi precedenti.

Questo metodo di verifica sarà prevalentemente adottato nelle fasi iniziali del progetto, in quanto i prodotti soggetti a verifica saranno generalmente meno complessi e, di conseguenza, l'operazione risulterà più agevole e meno dispendiosa in termini di risorse.
Le *attività*$_G$ in cui verrà applicata la tecnica del Walkthrough sono:

- **Revisione dei requisiti:** per assicurarsi che tutti i requisiti siano chiari, comprensibili e conformi alle esigenze del cliente;

- **Codifica del software:** per identificare errori di logica, pratiche non ottimali o potenziali problemi di manutenibilità nel codice;
- **Documentazione tecnica:** per assicurarsi che la documentazione tecnica sia accurata, chiara e completa.

3.2.3.3 Inspection

L'obiettivo di questa tecnica consiste nel rilevare eventuali difetti nel prodotto in esame mediante un'analisi mirata, piuttosto che tramite una revisione completa che non tende ad individuare dei difetti specifici.

Tale approccio prevede la specifica preventiva degli elementi da verificare, i quali vengono organizzati in liste di controllo. Queste liste fungono da *checklist* per valutare l'accuratezza dell'*attività*$_G$ d'ispezione, determinando se è stata condotta in modo corretto.L'inspection risulta particolarmente vantaggiosa quando ci si trova di fronte a documenti o codice complessi e strutturati, specialmente nelle fasi avanzate del progetto.
In questo contesto, il processo di verifica tramite inspection si rivela più efficace rispetto a un walkthrough, poiché quest'ultimo richiederebbe un considerevole impiego di risorse.

3.2.3.4 Analisi dinamica

Nel processo di sviluppo *software*$_G$, è essenziale condurre una verifica accurata del codice prodotto al fine di garantirne il corretto funzionamento. Questo processo si avvale dell'analisi dinamica, una categoria di metodologie di analisi che richiedono l'esecuzione effettiva del prodotto.

L'analisi dinamica coinvolge l'esecuzione di una serie di casi di *test*$_G$ durante l'attuazione del codice. L'obiettivo di tali *test*$_G$ è assicurare l'esecuzione accurata del *software*$_G$ e individuare eventuali discordanze tra i risultati ottenuti e quelli previsti.

È importante sottolineare che l'analisi dinamica non è applicabile alla documentazione. Per assicurare l'efficacia dell'analisi dinamica, è necessario automatizzare e rendere ripetibile questo processo, garantendo così una valutazione oggettiva del prodotto.
Nel contesto dell'ingegneria del *software*$_G$, il *test*$_G$ rappresenta la principale tecnica di analisi dinamica.

3.2.4 Testing

L'obiettivo del testing è assicurare il corretto funzionamento della componente soggetta al *test*$_G$, verificando che produca i risultati attesi e che rispetti accuratamente i vincoli assegnati. Questi *test*$_G$ sono inoltre strumentali per individuare eventuali anomalie di funzionamento.
Per ciascun *test*$_G$, è essenziale definire i seguenti parametri:

- **Ambiente:** il *sistema*$_G$ (hardware e *software*$_G$) utilizzato per eseguire il *test*$_G$;
- **Stato iniziale:** i parametri del *software*$_G$ al momento dell'esecuzione del *test*$_G$;
- **Input:** i dati forniti in entrata per l'esecuzione del *test*$_G$;
- **Output:** i risultati attesi in uscita in relazione a uno specifico input;
- **Commenti addizionali:** eventuali osservazioni o annotazioni aggiuntive pertinenti al *test*$_G$.

Tali *test*$_G$ andranno poi automatizzati.

3.2.4.1 Test di unità

Questi *test*$_G$ sono progettati per verificare singole unità di codice, come funzioni o metodi, in modo isolato e indipendente dal resto del *sistema*$_G$. L'obiettivo principale dei *test*$_G$ di unità è garantire che ogni unità di codice funzioni correttamente, conformandosi alle specifiche e restituendo i risultati attesi.
Tali *test*$_G$ si prestano quindi ad un alto grado di parallelismo, essi vengono pianificati durante la progettazione di dettaglio. Devono essere eseguiti per primi, in quanto verificano l'integrità e la correttezza della singola unità, prima dell'*integrazione*$_G$ con le altre.

Per implementare tali *test*$_G$, è consentito l'utilizzo di oggetti simulati o parziali, come mock e stub, al fine di separare l'unità di codice in esame dalle sue dipendenze esterne. Questo approccio permette di verificare il comportamento dell'unità in contesti controllati, garantendo un'efficace isolamento durante le fasi di *test*$_G$.

Un ulteriore obiettivo dei *test*$_G$ di unità consiste nel verificare la massima copertura possibile dei percorsi all'interno dell'unità. A tal fine, vengono appositamente progettati per attivare specifici percorsi, creando così una serie di *test*$_G$ dedicati che devono garantire una copertura completa del codice dell'unità, generando in tal modo la "structural coverage".

3.2.4.2 Test di integrazione

I *test*$_G$ di *integrazione*$_G$ sono una fase essenziale nell'analisi dinamica del *software*$_G$ e mirano a verificare il corretto funzionamento delle diverse unità di codice quando sono integrate per formare una componente più ampia o l'intero *sistema*$_G$.

Questa fase si concentra sull'interazione tra le parti del *software*$_G$ per garantire che lavorino sinergicamente secondo le specifiche del progetto.
I *test*$_G$ di *integrazione*$_G$ vengono pianificati durante la fase di progettazione architetturale e possono avere un approccio:

- **Top-down:** l'*integrazione*$_G$ inizia con le componenti di *sistema*$_G$ che presentano maggiori dipendenze e un maggiore valore esterno, consentendo la tempestiva disponibilità delle funzionalità di alto livello. Ciò consente di effettuare *test*$_G$ prolungati sulle funzionalità principali, rendendole disponibili inizialmente ma richiede molti mock;
- **Bottom-up:** l'*integrazione*$_G$ ha inizio dalle componenti di *sistema*$_G$ caratterizzate da minori dipendenze e un maggiore valore interno, ossia quelle meno visibili all'utente. Questo implica la necessità di meno mock ma ritarda il *test*$_G$ delle funzionalità utente esponendole per minor tempo a verifica.

Il team svolgerà, ove possibile, *test*$_G$ di *integrazione*$_G$ con l'approccio "Top down".

3.2.4.3 Test di sistema

Questi *test*$_G$ sono progettati per verificare l'intero *sistema*$_G$ *software*$_G$ rispetto ai requisiti specificati, garantendo che tutte le componenti siano integrate correttamente e che l'applicazione esegua le funzioni previste in modo accurato e affidabile.

In particolare devono essere implementati:

- **Test End-to-End:** coinvolgono l'esecuzione completa del *sistema*$_G$, dalla sua interfaccia utente fino alle componenti di backend, al fine di simulare l'esperienza completa dell'utente.

I *test*$_G$ di *sistema*$_G$ vengono eseguiti dopo che sono stati completati con successo i *test*$_G$ di *integrazione*$_G$.

3.2.4.4 Test di regressione

Mirano a garantire che le modifiche apportate al codice non abbiano introdotto nuovi difetti o compromesso le funzionalità preesistenti del *sistema*$_G$.
Questi *test*$_G$ sono essenziali per assicurare che le modifiche al *software*$_G$ non causino regressioni, ovvero la ricomparsa di errori precedentemente risolti o la compromissione di funzionalità precedentemente funzionanti.
I *test*$_G$ di regressione devono essere eseguiti ogni volta che vengono apportate modifiche al codice, garantendo una verifica continua e automatica della stabilità del *sistema*$_G$.

3.2.4.5 Test di accettazione

I *test*$_G$ di accettazione sono un passo fondamentale prima del rilascio del *software*$_G$ e sono progettati per garantire che il prodotto finale sia in grado di soddisfare le aspettative degli utenti finali e che risponda in modo appropriato ai requisiti specificati.

3.2.4.6 Sequenza delle fasi di test

La sequenza delle fasi di *test*$_G$ è la seguente:

1. Test di Unità;
2. Test di Integrazione;
3. Test di Regressione;
4. Test di Sistema;
5. Test di accettazione.

3.2.4.7 Codici dei test

Per classificare ogni *test*$_G$ che il team effettuerà durante l'*attività*$_G$ di verifica abbiamo deciso di associare un codice identificativo per ciascun *test*$_G$ nel formato:

T [tipo] [codice]

dove:

- **[tipo]:**
  - **U:** *test*$_G$ di unità;
  - **I:** *test*$_G$ di *integrazione*$_G$;
  - **S:** *test*$_G$ di *sistema*$_G$;
  - **R:** *test*$_G$ di regressione;
  - **A:** *test*$_G$ di accettazione.
- **[codice]:** è un numero associato al *test*$_G$ all'interno del suo tipo:
  - se il *test*$_G$ non ha un padre, è un semplice numero progressivo;
  - se il *test*$_G$ ha un padre, sarà nel formato:

    [codice.padre] . [codice.figlio]

    con:
    - **[codice.padre] :** identifica in maniera univoca il padre del *test*$_G$ all'interno della categoria di *test*$_G$ relativi al suo tipo;
    - **[codice.figlio] :** numero progressivo per identificare il *test*$_G$.

3.2.4.8 Stato dei test

Ad ogni *test*$_G$ sarà assegnato uno stato che rifletterà il risultato della sua esecuzione.
I risultati dei *test*$_G$ saranno registrati nel documento "*Piano di Qualifica*", all'interno della sezione "*Specifica dei test*".

Ogni *test*$_G$ potrà assumere uno dei seguenti stati:

- **NI:** il *test*$_G$ non è ancora stato implementato (Non implementato);
- **S:** il *test*$_G$ ha riportato esito positivo (Superato);
- **NS:** il *test*$_G$ ha riportato esito negativo (Non Superato).

3.2.4.9 Continuous Integration

L'*integrazione*$_G$ continua (CI) è una pratica di sviluppo *software*$_G$ che automatizza l'*integrazione*$_G$ frequente del codice sorgente da parte degli sviluppatori. Invece di attendere il completamento di grandi porzioni di lavoro o addirittura il rilascio finale, gli sviluppatori integrano il loro codice in un ramo principale condiviso più volte al giorno, solitamente dopo aver completato piccole modifiche.

Il processo di CI, che nel nostro progetto è implementato per la repo di *MVP*$_G$, è innescato dalla creazione di un Pull Request, quando uno dei programmatori vuole integrare le modifiche fatte con il ramo principale della repo. Tale Pull Request viene effettuata una volta che il programmatore ha finito di completare:

- **Implementazione di Feature**;
- **Creazione di Test**;
- **Refactoring**;
- **Risoluzione di Bug**.

Con la creazione di una Pull Request, avviene l'innesco di un applicazione di GitHub, chiamata Codacy, che esegue *test*$_G$ di tipo statico. Tale applicazione da esito positivo, nella pagina della Pull Request, se le modifiche fatte non hanno introdotto codice non conforme alle regole di analisi statica definite. Se l'applicazione da un esito di fallimento, il programmatore deve cambiare le modifiche finché il codice passa l'analisi statica.

Alla conclusione della Codacy, tramite i *workflow*$_G$ di GitHub, il codice del prodotto viene ottenuto e inizia la parte di build del prodotto. Tramite il comando:

```
    \textit{docker}\textsubscript{\textit{G}} compose up
```

viene effettuato il pull da DockerHub di tutte le immagini necessarie per costruire tutti in componenti del prodotto necessari al testing.

Una volta conclusa la costruzione dell prodotto con successo inizia la parte di testing statica e dinamica grazie al profilo di testing dell'ambiente *Docker*$_G$. Prima di tutto, iniziano i *test*$_G$ statici usando Pylint. L'esito di tali *test*$_G$ viene esposto nel file *README.md* usando un badge di GitHub che mostra un punteggio compreso tra 0 e 10 rappresentante la qualita del codice.

Una volta finiti i *test*$_G$ statici iniziano i *test*$_G$ dinamici che sono:

- **Test di Unita**;
- **Test di Integrazione**;
- **Test di Sistema**.

Se almeno uno dei *test*$_G$ fallisce, allora il *workflow*$_G$ si ferma con stato di fallimento, manda una notifica all programmatore che ha creato la Pull Request e chiude la Pull Request. Se tutti i *test*$_G$ vengono eseguiti con successo, sarà possibile apportare le modifiche fatte nel ramo principale.

Alla conclusione dei *test*$_G$ viene riportato il code coverage in un file chiamato coverage.xml. Questo file grazie a GitHub action viene elaborato e i risultati vengono trasformati in badge di GitHub esposti nel file *README.md* della *repository*$_G$.

3.2.5 Strumenti

- **Spell checker:** controllo ortografico integrato nell'ambiente di lavoro;
- **Modulo "Pytest" Python:** libreria *standard*$_G$ che offre una serie di funzionalità per creare, organizzare e eseguire *test*$_G$ di unità in modo efficace e automatizzato;
- **Modulo "Unittest" Python:** libreria *standard*$_G$ che offre una serie di funzionalità per creare, organizzare e eseguire *test*$_G$ di unità;
- **Modulo "Pylint" Python:** libreria open source che analizza il codice sorgente *Python*$_G$ per identificare potenziali errori, bug e cattive abitudini di programmazione;
- **Applicazione GitHub Codacy:** *servizio*$_G$ che verifica il codice tramite l'analisi statica;
- **Coverallas:** *servizio*$_G$ che traccia e mostra il code coverage;
- **Codecov:** *servizio*$_G$ che traccia e mostra il code coverage;

3.3 Validazione

3.3.1 Introduzione

Nello *standard*$_G$ ISO/IEC 12207:1995, il processo di validazione è definito come il processo di conferma attraverso dimostrazione oggettiva che i requisiti specificati per un'operazione o un *sistema*$_G$ siano stati soddisfatti. In altre parole, la validazione è il processo finalizzato a garantire che il prodotto *software*$_G$ risponda alle esigenze e alle aspettative degli utenti finali, dimostrandolo attraverso prove, *test*$_G$ o altri metodi oggettivi.

Questo processo è fondamentale per assicurarsi che il *software*$_G$ sviluppato corrisponda agli obiettivi iniziali del progetto. Un aspetto cruciale della validazione è l'interazione diretta con il *committente*$_G$ e il *proponente*$_G$ al fine di ottenere un feedback immediato e garantire un chiaro allineamento tra ciò che è stato sviluppato e le aspettative degli utenti finali.

Il prodotto finale deve quindi:

- soddisfare tutti i requisiti concordati con il *proponente*$_G$;
- poter essere eseguito correttamente nel suo ambiente di utilizzo finale.

L'obiettivo è giungere a un prodotto finale pronto per il rilascio, segnando così la conclusione del ciclo di vita del progetto didattico.

3.3.2 Procedura di validazione

Il processo di validazione riceverà in ingresso i *test*$_G$ elencati nella sezione 3.2.4 (*Testing*).
In particolare, il *test*$_G$ di accettazione è considerato il nucleo essenziale di questo processo, finalizzato a garantire la validazione del prodotto.
I *test*$_G$ considerati dovranno valutare:

- Soddisfacimento dei casi d'uso;
- Soddisfacimento dei requisiti obbligatori;
- Soddisfacimento di altri requisiti concordati con il *committente*$_G$.

3.3.3 Strumenti

- Strumenti utilizzati per la *Verifica*.

3.4 Gestione della configurazione

3.4.1 Introduzione

La gestione della configurazione del progetto è un processo che norma il tracciamento e il controllo delle modifiche a documenti e prodotti del *software*$_G$ detti Configuration Item (CI).
La gestione della configurazione viene applicata a qualunque categoria di "artefatti" coinvolti nel ciclo di vita del *software*$_G$.
Secondo lo *standard*$_G$ ISO/IEC 12207:1995, la gestione della configurazione del *software*$_G$ è un processo di identificazione, organizzazione e controllo delle modifiche apportate ai prodotti *software*$_G$ durante il loro ciclo di vita.
Lo *standard*$_G$ IEEE 828-2012 definisce la gestione della configurazione del *software*$_G$ come "un processo disciplinato per gestire l'evoluzione del *software*$_G$".

3.4.2 Numeri di versionamento

La convenzione per identificare la versione di un documento è nel formato X.Y.Z con:

- **X:** viene incrementato al raggiungimento di *RTB*$_G$, *PB*$_G$ ed eventualmente CA;
- **Y:** viene incrementato quando vengono apportate modifiche significative al documento, come cambiamenti strutturali, nuove sezioni importanti o modifiche sostanziali nel contenuto;
- **Z:** viene incrementato per modifiche minori o aggiornamenti al documento. Questi potrebbero includere correzioni di errori, miglioramenti marginali o l'aggiunta di nuovi contenuti meno rilevanti.

L'incremento dei valori più significativi porta i meno significativi a zero. Ad esempio, se in un documento nella versione 0.6.4 vengono effettuate modifiche significative e viene conseguentemente incrementato il valore Y, si passa alla versione 0.7.0 e non alla versione 0.7.4.
Ogni variazione di versione deve essere presente nel registro delle modifiche.

3.4.3 Repository

Di seguito sono elencate le *repository*$_G$ del team ByteOps con i relativi riferimenti:

- Sorgente-documenti: *repository*$_G$ per il versionamento, la gestione e lo sviluppo dei file sorgente relativi alla documentazione;
- Documents: *repository*$_G$ destinata ai committenti/proponenti, dove vengono esclusivamente condivisi i file in formato PDF relativi alla documentazione, ottenuti dalla compilazione dei file sorgente presenti nella cartella "Sorgente-documenti";

- proof-of-concept: *repository*$_G$ destinata al *POC*$_G$.

3.4.3.1 Struttura repository

I *repository*$_G$ destinati alla documentazione sono organizzati come segue:

- **Candidatura:** contenente i documenti richiesti per la candidatura;
  - **Verbali:** contiene tutti i verbali redatti nel corso del periodo di candidatura, distinti tra verbali esterni e interni;
  - **Lettera di presentazione**;
  - **Valutazione dei costi e assunzione impegni**;
  - **Valutazione dei capitolati**.
- **RTB:** contenente i documenti richiesti per la revisione omonima;
  - **Verbali:** contiene tutti i verbali redatti nel corso del periodo della *RTB*$_G$, distinti tra verbali esterni e interni;
  - **Piano di Qualifica**;
  - **Piano di Progetto**;
  - **Analisi dei Requisiti**;
  - **Glossario**;
  - **Norme di Progetto**.
- **PB**.

3.4.4 Sincronizzazione e Branching

3.4.4.1 Documentazione

Il processo operativo utilizzato per la redazione della documentazione, noto come "Feature Branch workflow", implica la creazione di un ramo dedicato per ciascun documento o sezione da elaborare. Tale metodologia permette una parallelizzazione agile dei lavori evitando sovrascritture indesiderate di altri lavori e permette l'adozione dei principi "Documentation as code" definiti nella *sezione 3.1.2*.

Convenzioni per la nomenclatura dei branch relativi alle attività di redazione o modifica di documenti

- Il nome del *branch*$_G$ deve presentare l'identificativo del documento che si vuole redarre o modificare.
  Ad ogni documento è associato un identificativo, come descritto nella seguente tabella:

Documento	ID
Verbale interno	VI
Verbale esterno	VE
Norme di Progetto	NdP
Piano di Qualifica	PdQ
Piano di Progetto	PdP
Analisi dei Requisiti	AdR

Table 5: ID per la nomenclatura dei branch relativi alla documentazione

- Nel caso dei verbali, dopo l'identificativo del documento si aggiunge un underscore seguito dalla data:
  `IdDocumento_dd-mm-yyyy` (ex. VI_27-12-2023);
- Nel caso della redazione di una specifica sezione di un documento, il nome del *branch*$_G$ deve avere il formato:
  `IdDocumento_NomeSezione` (ex. NdP_Documentazione);
- Nel caso di modifica di una specifica sezione di un documento, il nome del *branch*$_G$ deve avere il formato:
  `IdDocumento_ModNomeSezione` (ex. NdP_ModDocumentazione).

3.4.4.2 Sviluppo

Gitflow è lo stile di flusso di lavoro Git che utilizza il team ByteOps per lo sviluppo.

Flusso generale di Gitflow

1. **Branch** `develop`**:** viene creato a partire dal *branch*$_G$ principale (`main`). È il punto di partenza per lo sviluppo di nuove funzionalità;
2. **Branch** `release`**:** creato da `develop`, questo *branch*$_G$ gestisce la preparazione del *software*$_G$ per un rilascio. Durante questa fase, sono consentite solo correzioni di bug e miglioramenti minori;
3. **Branch** `feature`**:** creati da `develop`, sono utilizzati per lo sviluppo di nuove funzionalità o miglioramenti;
4. **Merge di** `feature` **in** `develop`**:** quando una funzionalità è completa, il *branch*$_G$ `feature` viene "fuso" nel *branch*$_G$ `develop`;

5. **Merge di `release` in `develop` e `main`:** dopo il completamento del *branch*$_G$ `release`, quest'ultimo viene unito sia in `develop` che in `main`, segnalando un nuovo rilascio stabile;
6. **Branch `hotfix`:** creato da `main` in caso di problemi critici rilevati nell'ambiente di produzione;
7. **Merge di `hotfix` in `develop` e `main`:** una volta risolto il problema, il *branch*$_G$ `hotfix` viene unito sia in `develop` che in `main` per garantire coerenza tra le versioni.

Comandi di comodo

- Inizializzare Gitflow

```
git flow init
```

- Sviluppo di una Nuova Funzionalità

```
git flow feature start nome_feature
```

```
git flow feature finish nome_feature
```

- Risoluzione di un bug

```
git flow hotfix start nome_hotfix
```

```
git flow hotfix finish nome_hotfix
```

- Rilascio di una nuova versione

```
git flow release start X.X.X
```

```
git flow release finish X.X.X
```

- Pubblicazione delle modifiche

```
git push origin develop
git push origin master
git push origin --tags
```

```
git push origin feature/nome_feature
git push origin hotfix/nome_hotfix
git push origin release/X.X.X
```

3.4.4.3 Pull Request

Dopo aver portato a termine le *attività*$_G$ nel proprio *branch*$_G$, il responsabile del suo sviluppo è tenuto ad avviare una Pull Request per incorporare le modifiche effettuate nel ramo principale (main). La Pull Request viene approvata solo dopo che le modifiche sono state verificate sulla base di quanto descritto nella *sezione 3.2*.

Procedura per la creazione di Pull Request

Per creare una Pull Request eseguire i seguenti passaggi:

1. Accedere al *repository*$_G$ GitHub e cliccare sulla scheda "Pull requests";
2. Cliccare il pulsante "New Pull Request";
3. Selezionare il *branch*$_G$ di partenza ed il *branch*$_G$ target;
4. Cliccare il pulsante "Create Pull Request";
5. Aggiungere un titolo alla Pull Request;

6. Aggiungere una descrizione;
7. Selezionare i verificatori;
8. Come convenzione, l'assegnatario è colui che richiede la Pull Request;
9. Selezionare le *label*$_G$;
10. Selezionare il progetto;
11. Selezionare le *milestone*$_G$;
12. Clicca il pulsante "Create Pull Request";
13. Nel caso siano presenti conflitti seguire le istruzioni in *Github*$_G$ per rimuovere tali conflitti.

3.4.5 Controllo di configurazione

3.4.5.1 Change request (Richiesta di modifica)

Seguendo lo *standard*$_G$ ISO/IEC 12207:1995 per affrontare questo processo in modo strutturato le *attività*$_G$ sono:

1. **Identificazione e registrazione**
   Le change request vengono identificate, registrate e documentate accuratamente. Questo include informazioni come la natura della modifica richiesta, l'urgenza e l'impatto sul *sistema*$_G$ esistente.
   L'identificazione avviene tramite la creazione di una *issue*$_G$ nell'ITS con *label*$_G$: "Change request";
2. **Valutazione e analisi**
   Le change request vengono valutate dal team per determinare la loro fattibilità, importanza e impatto sul progetto. Si analizzano i costi e i benefici associati all'implementazione della modifica;
3. **Approvazione o rifiuto**
   Il responsabile valuta le informazioni raccolte e decide se approvare o respingere la change request. Questa decisione può essere basata su criteri come il budget, il tempo, le priorità degli *stakeholder*$_G$;
4. **Pianificazione delle modifiche**
   Se una change request viene approvata, viene pianificata e integrata nel ciclo di sviluppo del *software*$_G$. Questo può richiedere, ad esempio, la rinegoziazione dei tempi di consegna e la revisione del piano di progetto;

5. **Implementazione delle modifiche**
   Le modifiche vengono effettivamente implementate. Durante questo processo, è fondamentale mantenere una traccia accurata di ciò che viene fatto per consentire una corretta documentazione e, se necessario, la possibilità di un *rollback*$_G$;

6. **Verifica e validazione**
   Le modifiche apportate vengono verificate per assicurarsi che abbiano raggiunto gli obiettivi previsti e non abbiano introdotto nuovi problemi o errori;

7. **Documentazione**
   Tutti i passaggi del processo di gestione delle change request vengono documentati accuratamente per garantire la trasparenza e la tracciabilità. Questa documentazione è utile per futuri riferimenti e per l'apprendimento delle modifiche apportate;

8. **Comunicazione agli stakeholder**
   Durante tutto il processo è importante comunicare in modo chiaro e tempestivo con gli *stakeholder*$_G$, per mantenere trasparenza e fiducia.

3.4.6 Configuration Status Accounting (Contabilità dello Stato di Configurazione)

La Contabilità dello Stato di Configurazione è un'*attività*$_G$ dedicata a registrare e monitorare lo stato di tutte le configurazioni di un *sistema*$_G$ *software*$_G$ durante il suo ciclo di vita.
Questo processo è cruciale per mantenere la trasparenza e la tracciabilità nel ciclo di vita del *software*$_G$, aiutando a gestire le configurazioni in modo uniforme e a mantenere un registro accurato di tutte le *attività*$_G$ e le modifiche relative agli elementi di configurazione.

- ***Registrazione delle configurazioni***: registrazione delle informazioni dettagliate su ogni Configuration Item;
  - **Documentazione:** le informazioni relative alla configurazione sono presenti nella prima pagina di ciascuno;
  - **Sviluppo:** le informazioni relative alla configurazione sono inserite come prime righe di ciascun file sotto forma di commento.
- **Stato e cambiamenti**: tenere traccia dello stato attuale di ciascun elemento di configurazione e di tutti i cambiamenti che avvengono nel corso del tempo.
  Ciò include le versioni attuali, le revisioni, le modifiche e le baseline;
  - **Registro delle modifiche:** per monitorare lo stato di ciascun Configuration Item si utilizza il registro delle modifiche incorporato in ognuno di essi, come specificato in "*registrazione delle configurazioni*";

  - **Branching & DashBoard:** per verificare se vi sono *attività*$_G$ in corso su un Configuration Item, è sufficiente controllare se ci sono *branch*$_G$ attivi correlati e, dato che ogni *issue*$_G$ è associata a un CI tramite *label*$_G$, verificare le eventuali *issue*$_G$ contrassegnate come "In Progress" nella colonna corrispondente della Dashboard del progetto.

- **Supporto per la gestione delle change request**: registra e documenta le modifiche apportate agli elementi di configurazione in risposta alle richieste di modifica.
  Per gestire le richieste di modifica, si utilizza l'Issue Tracking System (ITS) di GitHub, creando una *issue*$_G$ contrassegnata con l'etichetta "Change request".

Per approfondimenti su come creare issues e associare labels a una *issue*$_G$, consultare la sezione relativa al *Ticketing*.

3.4.7 Release management and delivery

Nello *standard*$_G$ ISO/IEC 12207:1995, il processo "Release management and delivery" è definito come un processo che si occupa della pianificazione, del coordinamento e del controllo delle *attività*$_G$ necessarie per preparare e distribuire una versione di un prodotto *software*$_G$ per l'uso operativo.
Questo processo comprende la gestione delle release, la distribuzione del *software*$_G$, la preparazione della documentazione correlata e altre *attività*$_G$ correlate al rilascio e alla consegna del prodotto *software*$_G$.

Il team ByteOps ha stabilito che la creazione di una release deve avvenire in relazione alle baseline stabilite per il progetto didattico, ovvero *RTB*$_G$ (Requirements and Technology Baseline), *PB*$_G$ (Product Baseline) ed eventualmente CA (Customer Acceptance). Questo processo inoltre deve essere attuato solo dopo una completa verifica e validazione del prodotto seguendo le procedure dettagliate nelle *sezioni Verifica(3.2)* e *Validazione(3.3)*.

In conformità con quanto definito nella *sezione 3.4.2*, relativa al versionamento dei numeri, ogni nuova release comporta un incremento di 1 nell'elemento numerico X del formato X.Y.Z.

3.4.7.1 Procedura per la creazione di una release

Per la creazione di una release eseguire la seguente procedura:

- Accedi alla *repository*$_G$ GitHub;
- Accedi alla sezione "Releases";
- Clicca sul pulsante "Create a new release";

- Clicca sul pulsante "Choose a tag" e selezionare il tag che identifica la versione della release. Nel caso il tag desiderato non fosse presente nella lista, è possibile crearne uno nuovo scrivendo la versione desiderata e cliccando sul pulsante "Create new tag". Per la corretta definizione della versione vedi la *sezione 3.4.2* relativa ai numeri di versionamento;
- Seleziona il *branch*$_G$ main come *branch*$_G$ target di cui si vuole creare una release;
- Scegliere un titolo per la release. La convenzione adottata è quella di inserire come titolo il nome della baseline per la quale si vuole creare la release.
- Scrivi una breve descrizione per la release;
- Clicca "Publish release".

Dopo aver seguito la procedura, nella sezione "Releases" di GitHub sarà visibile la release appena creata e sarà possibile scaricare il file.zip relativo.

3.4.8 Strumenti

Le tecnologie adottate per la gestione dei Configuration Item sono:

- **Git**: Version Control System distribuito utilizzato per il versionamento dei Configuration Item;
- **GitHub**: *piattaforma*$_G$ web per il controllo di versione (tramite Git) dei Configuration Item e per il *Ticketing*. È impiegata per gestire le richieste di modifica tramite *issue*$_G$ e *label*$_G$, oltre che per la contabilità dello stato di configurazione.

3.5 Joint review

3.5.1 Introduzione

Il processo di revisione congiunta costituisce un metodo formale per valutare lo stato e i risultati di un'*attività*$_G$ all'interno di un progetto, coinvolgendo sia il livello gestionale che tecnico. Tale procedura viene attuata durante l'intero periodo contrattuale.
Può essere attivata da due entità qualsiasi, di cui una (la parte recensita) esamina criticamente l'altra parte (la parte recensente).

Nel nostro caso i recensori sono gli *stakeholder*$_G$: *committente*$_G$, *proponente*$_G$ mentre noi fornitori siamo i recensiti.

Nelle sezioni successive saranno descritte in dettaglio le seguenti *attività*$_G$ che fanno parte del processo di Joint Review:

- Implementazione del processo;
- Project management reviews;
- Revisioni tecniche;

3.5.2 Implementazione del processo

Questa *attività*$_G$ comprende i seguenti compiti:

3.5.2.1 Revisioni periodiche

Saranno condotte revisioni periodiche in corrispondenza di *milestone*$_G$ prestabilite come specificato nel documento *Piano di Progetto*.

3.5.2.2 SAL

Ogni due settimane si tiene una revisione *SAL*$_G$ (Stato Avanzamento Lavori) tra il *fornitore*$_G$ e il *proponente*$_G$. Lo scopo è valutare il lavoro svolto nelle due settimane successive alla revisione *SAL*$_G$ precedente, per determinare se le aspettative e le scadenze sono state rispettate e se sono emerse difficoltà. Inoltre, durante ogni *SAL*$_G$, si pianificano le successive task da svolgere.

3.5.2.3 Revisioni ad hoc

È prevista la possibilità di programmare revisioni ad hoc nel caso in cui uno qualsiasi degli *stakeholder*$_G$ le ritenga necessarie. Durante tali revisioni, le parti coinvolte valutano dettagliatamente lo stato avanzamento dei lavori, discutono eventuali problematiche emerse e definiscono le azioni correttive necessarie.

3.5.2.4 Risorse per le revisioni

Tutte le risorse necessarie per condurre le revisioni sono concordate tra le parti. Queste risorse includono personale, strutture, hardware, *software*$_G$ e strumenti.

3.5.2.5 Elementi da concordare

Le parti concordano i seguenti elementi in ciascuna revisione:

- Agenda della riunione;
- Prodotti *software*$_G$ (risultati di un'*attività*$_G$) e relativi problemi da esaminare;
- Ambito e procedure;
- Criteri di ingresso e uscita per la revisione.

3.5.2.6 Documentazione e distribuzione dei risultati

I risultati della revisione devono essere documentati e distribuiti tramite i *Verbali Esterni*. La parte recensente riconoscerà alla parte recensita l'adeguatezza (ad esempio approvazione, disapprovazione o approvazione condizionale) dei risultati della revisione.

3.5.3 Project management reviews

3.5.3.1 Introduzione

Nello *standard*$_G$ ISO/IEC 12207:1995, l'*attività*$_G$ di "project management reviews" si riferisce a un processo di revisione e valutazione dei progetti *software*$_G$ in corso o completati.

Questa *attività*$_G$ è volta a garantire che il progetto venga eseguito in modo efficiente e conforme agli obiettivi e ai requisiti definiti. Le project management reviews sono condotte periodicamente durante l'intero ciclo di vita del progetto e coinvolgono tipicamente il team di gestione del progetto, i responsabili delle diverse aree funzionali coinvolte nel progetto e altri *stakeholder*$_G$ pertinenti.

3.5.3.2 Stato del Progetto

Lo scopo è quello di verificare se il progetto è in linea con i piani stabiliti in termini di budget, tempistiche e obiettivi di qualità. L'esito della revisione è discusso tra gli *stakeholder*$_G$ e prevede quanto segue:

1. Garantire che le *attività*$_G$ progrediscano secondo i piani, basandosi su una valutazione dello stato dell'*attività*$_G$ e/o del prodotto *software*$_G$;
2. Mantenere il controllo globale del progetto attraverso l'allocazione adeguata delle risorse;
3. Modificare la direzione del progetto o determinare la necessità di pianificazioni alternative;
4. Valutare e gestire le questioni legate al rischio che potrebbero compromettere il successo del progetto.

3.5.4 Revisioni Tecniche

Le revisioni tecniche devono essere condotte per valutare i prodotti o servizi *software*$_G$ presi in considerazione e fornire evidenze che:

1. Siano completi;
2. Siano conformi agli *standard*$_G$ e alle specifiche;

3. Le modifiche ad essi siano correttamente implementate e influiscano solo sulle aree identificate dalla change request;
4. Siano conformi agli schemi temporali applicabili;
5. Siano pronti per la successiva *attività*$_G$;
6. Lo sviluppo, l'operatività o la manutenzione siano condotti secondo i piani, gli schemi temporali, gli *standard*$_G$ e le linee guida del progetto.

3.5.5 Strumenti

- **Zoom:** per revisioni tecniche con i committenti;
- **Google Meet:** per *SAL*$_G$ con la *proponente*$_G$;
- **Element:** per richieste di revisioni ad hoc alla *proponente*$_G$.

3.6 Risoluzione dei problemi

3.6.1 Introduzione

Il processo di risoluzione dei problemi è un processo che mira ad analizzare e risolvere i problemi (incluse le non conformità) di qualunque natura o fonte, che vengono rilevati durante l'esecuzione di sviluppo, manutenzione o altri *processi*$_G$.

L'obiettivo è fornire un mezzo tempestivo, responsabile e documentato per garantire che tutti i problemi scoperti siano analizzati e risolti e che siano riconosciute le tendenze.
Il processo di risoluzione dei problemi mira ad analizzare e risolvere i problemi, inclusi i casi di non conformità, che emergono durante tutto il ciclo di vita del prodotto. L'obiettivo principale è fornire un approccio tempestivo, responsabile e documentato per affrontare in modo efficace ogni problematica individuata.

Questo processo non si limita a risolvere i problemi immediati, ma si propone anche di identificarne le cause profonde e di adottare misure preventive per evitare che si verifichino nuovamente in futuro. Inoltre, mira a promuovere un ambiente di miglioramento continuo, dove l'apprendimento dagli errori passati gioca un ruolo cruciale nella crescita e nell'ottimizzazione dei *processi*$_G$.

Un aspetto importante della risoluzione dei problemi è l'adozione di approcci strutturati e metodologie efficaci, che includono la raccolta di dati, l'analisi delle cause alla radice, la valutazione degli impatti e la definizione di azioni correttive e preventive.

È essenziale anche mantenere una documentazione accurata di tutti i problemi identificati e delle relative soluzioni implementate, al fine di garantire la trasparenza, la tracciabilità e la possibilità di revisione nel tempo.

3.6.2 Gestione dei rischi

Nel documento *Piano di Progetto*, nella sezione "Analisi dei rischi", vengono identificati dal responsabile tutti i potenziali rischi di progetto, inclusa la probabilità della loro occorrenza e le misure di mitigazione. Per ogni fase di avanzamento, è dedicata una sezione alla documentazione dei problemi riscontrati, con un'analisi del loro impatto e una valutazione dell'esito della mitigazione programmata. Un esito negativo evidenzia una mitigazione inadeguata che richiede modifiche.

3.6.2.1 Codifica dei rischi

La convenzione utilizzata per la codifica dei rischi è la seguente:

`R[Tipologia]-[Probabilità][Priorità]-[Indice]` : *Nome associato al rischio*

Tipologia:
Natura del rischio:

- **T**: Rischi legati all'utilizzo delle tecnologie;
- **O**: Rischi legati all'organizzazione del gruppo;
- **P**: Rischi legati agli impegni personali dei membri del gruppo.

Probabilità:
Valore alfabetico che indica la probabilità di occorrenza del rischio:

- **1**: Alta;
- **2**: Media;
- **3**: Bassa.

Priorità:
Valore numerico che indica la pericolosità del rischio:

- **A**: Alta;
- **M**: Media;
- **B**: Bassa.

Indice: Valore numerico incrementale che determina univocamente il rischio relativamente ad una specifica tipologia.

3.6.2.2 Metriche

Metrica	Nome	Riferimento
M11RNP	Rischi non previsti (RNP)	M11RNP

Table 6: Metriche relative alla gestione dei processi

3.6.3 Identificazione dei problemi

Nel caso in cui un membro del team identifichi un problema, è obbligatorio notificarlo immediatamente al gruppo, e contemporaneamente, deve essere aperta una segnalazione nel *sistema*$_G$ di tracciamento delle *issue*$_G$ (ITS) con l'etichetta "bug" e una descrizione completa del problema. La procedura per l'apertura di una *issue*$_G$ è descritta nel dettaglio nella sezione *Ticketing*.

3.6.4 Strumenti

- **GitHub:** per la segnalazione delle Issue.

3.7 Gestione della qualità

3.7.1 Introduzione

Le *attività*$_G$ correlate al processo di gestione della qualità mirano a garantire la qualità del flusso operativo adottato dal *fornitore*$_G$ sui prodotti sviluppati, al fine di soddisfare pienamente le aspettative del cliente e del *proponente*$_G$, nonché di rispettare i requisiti di qualità specificati. Questo include la definizione degli obiettivi di qualità, l'identificazione delle metriche e dei criteri di qualità, la pianificazione e l'esecuzione delle *attività*$_G$ di controllo della qualità, e la verifica della qualità attraverso revisioni, ispezioni e *test*$_G$. L'obiettivo è garantire che il prodotto *software*$_G$ sia conforme alle aspettative degli utenti e ai requisiti del progetto.

La gestione della qualità rappresenta quindi un approccio *olistico*$_G$ che abbraccia l'intero ciclo di vita del *software*$_G$, dal concepimento all'implementazione e oltre, con l'obiettivo di assicurare che il prodotto finale rispetti gli *standard*$_G$ di qualità predefiniti e consenta un continuo miglioramento dei *processi*$_G$.

3.7.2 Attività

Le *attività*$_G$ che il team si impegna a svolgere per assicurare qualità dei *processi*$_G$ e di conseguenza dei prodotti sono:

1. **Definizione degli Standard di qualità:** la gestione della qualità inizia con la definizione chiara degli *standard*$_G$ di qualità che il *software*$_G$ dovrebbe raggiungere. Questi *standard*$_G$ possono includere requisiti funzionali e non funzionali;
2. **Pianificazione della qualità:** viene sviluppato un piano di qualità che identifica *attività*$_G$ specifiche, risorse e tempistiche per garantire la qualità del prodotto durante l'intero ciclo di vita del progetto;
3. **Assicurazione della qualità:** la fase di assicurazione della qualità coinvolge *attività*$_G$ continue di monitoraggio e valutazione per garantire che i *processi*$_G$ siano conformi agli *standard*$_G$ di qualità stabiliti;
4. **Controllo della qualità:** il controllo della qualità include l'esecuzione di *test*$_G$ e verifiche per garantire che il prodotto soddisfi gli *standard*$_G$ di qualità e risponda alle specifiche richieste;
5. **Gestione delle modifiche:** un *sistema*$_G$ di gestione delle modifiche è implementato per gestire e controllare le modifiche al prodotto. Questo assicura che ogni modifica venga valutata in termini di impatto sulla qualità complessiva del prodotto;
6. **Miglioramento continuo e correzione:** la gestione della qualità promuove il miglioramento continuo dei *processi*$_G$. Attraverso la raccolta di feedback, l'analisi delle prestazioni e l'implementazione di *best practice*$_G$, il team cerca costantemente di ottimizzare la qualità dei *processi*$_G$ e dei prodotti;
7. **Coinvolgimento degli stakeholder:** gli *stakeholder*$_G$, inclusi clienti e utenti finali, sono coinvolti nel processo di gestione della qualità. I loro feedback sono preziosi per garantire che il prodotto risponda alle esigenze e alle aspettative;
8. **Formazione e competenza del team:** la formazione continua del team è essenziale per mantenere elevate competenze e conoscenze. Un team ben addestrato è in grado di produrre un prodotto di alta qualità.

3.7.3 Piano di Qualifica

Le *attività*$_G$ di pianificazione, assicurazione e controllo della qualità sono ampiamente trattate nel documento *Piano di Qualifica*, il quale riveste un ruolo essenziale nel contesto del processo di "gestione della qualità", offrendo un contributo significativo per assicurare che il prodotto soddisfi gli *standard*$_G$ di qualità richiesti. In tale documento vengono definite in dettaglio le specifiche di qualità relative al prodotto, identificando le azioni di controllo necessarie per garantire il rispetto di tali specifiche e assegnando le responsabilità pertinenti per l'esecuzione di tali azioni.

3.7.4 PDCA

Nell'ambito dell'*attività*$_G$ 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*$_G$ e dei prodotti. Affinché si possa ottenere un miglioramento effettivo, è cruciale attuare scrupolosamente le seguenti 4 fasi:

- **Plan**
  Consiste nello svolgere le *attività*$_G$ di pianificazione necessarie per stabilire quali *processi*$_G$ 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*$_G$;
- **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.

3.7.5 Strumenti

Gli strumenti impiegati per la gestione della qualità sono rappresentati dalle metriche.

3.7.6 Struttura e identificazione metriche

- **Codice:**
  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;
- **Scopo:** il motivo per cui è importante tale misura al fine del progetto.

Eventualmente anche:

- **Formula:** come viene calcolata la metrica;
- **Strumento:** lo strumento che viene usato per calcolare la metrica.

3.7.7 Criteri di accettazione

Per ciascuna metrica nel documento *Piano di Qualifica* vengono definiti in formato tabellare:

- **Valore accettabile:** valore che la metrica deve raggiungere per essere considerata soddisfacente o confrome agli *standard*$_G$ stabiliti;
- **Valore preferibile:** valore ideale che dovrebbe essere assunto dalla metrica.

3.7.8 Metriche

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

Table 7: Metriche relative alla gestione della qualità

4 Processi organizzativi

Lo sviluppo *software*$_G$ è un ambito complesso e multidisciplinare che richiede una pianificazione accurata, una gestione efficiente delle risorse e un controllo rigoroso della qualità. L'adozione di *processi*$_G$ organizzativi ben strutturati diventa quindi cruciale per garantire il successo di un progetto *software*$_G$.

4.1 Gestione dei Processi

4.1.1 Introduzione

La Gestione dei Processi si occupa di stabilire, implementare e migliorare i *processi*$_G$ che guidano la realizzazione del *software*$_G$, al fine di raggiungere gli obiettivi prefissati e soddisfare le esigenze degli *stakeholder*$_G$.

Le *attività*$_G$ di gestione di processo sono:

1. **Definizione dei Processi:**
    - Identificare e documentare i *processi*$_G$ chiave coinvolti nello sviluppo *software*$_G$;
    - Stabilire linee guida e procedure per l'esecuzione di ciascun processo.
2. **Pianificazione e Monitoraggio:**
    - Elaborare piani dettagliati per l'esecuzione dei *processi*$_G$;
    - Monitorare costantemente l'avanzamento, l'efficacia e la conformità ai requisiti pianificati;
    - Stimare i tempi, le risorse ed i costi.

3. **Valutazione e Miglioramento Continuo:**
    - Condurre valutazioni periodiche dei *processi*$_G$ per identificare aree di miglioramento;
    - Implementare azioni correttive e preventive per ottimizzare i *processi*$_G$.
4. **Formazione e Competenze:**
    - Assicurare che il personale coinvolto nei *processi*$_G$ sia adeguatamente formato;
    - Mantenere e sviluppare le competenze necessarie per l'efficace gestione dei *processi*$_G$.
5. **Gestione dei Rischi:**
    - Identificare e valutare i rischi associati ai *processi*$_G$;
    - Definire strategie per mitigare o gestire i rischi identificati.

4.1.2 Pianificazione

4.1.2.1 Descrizione

La pianificazione riveste un ruolo centrale nella gestione dei *processi*$_G$, poiché mira a creare un piano organizzato e coerente per assicurare un'efficace esecuzione delle *attività*$_G$ durante l'intero ciclo di vita del *software*$_G$.

Il responsabile del progetto assume il compito di coordinare ogni aspetto della pianificazione delle *attività*$_G$, che include l'allocazione delle risorse, la definizione dei tempi e la redazione di piani dettagliati. Inoltre, il responsabile si assicura che il piano elaborato sia fattibile e possa essere eseguito correttamente ed efficientemente dai membri del team.

I piani associati all'esecuzione del processo devono comprendere descrizioni dettagliate delle *attività*$_G$ e delle risorse necessarie, specificando le tempistiche, le tecnologie impiegate, le infrastrutture coinvolte e il personale assegnato.

4.1.2.2 Obiettivi

L'obiettivo primario della pianificazione è assicurare che ciascun membro del team assuma ogni ruolo almeno una volta durante lo svolgimento del progetto, promuovendo così una distribuzione equa delle responsabilità e un arricchimento delle competenze all'interno del team.

La pianificazione, stilata dal responsabile, è integrata nel documento del *Piano di Progetto*. Questo documento fornisce una descrizione completa delle *attività*$_G$ e dei compiti necessari per raggiungere gli obiettivi prefissati in ogni periodo del progetto.

4.1.2.3 Assegnazione dei ruoli

Durante l'intero periodo del progetto, i membri del gruppo assumeranno sei ruoli distinti, ovvero assumeranno le responsabilità e svolgeranno le mansioni tipiche dei professionisti nel campo dello sviluppo *software*$_G$.
Nei successivi paragrafi sono descritti in dettaglio i seguenti ruoli:

- Responsabile;
- Amministratore;
- Analista;
- Progettista;
- Programmatore;
- Verificatore.

4.1.2.4 Responsabile

Figura fondamentale che coordina il gruppo, fungendo da punto di riferimento per il *committente*$_G$ e il team, svolgendo il ruolo di mediatore tra le due parti.
In particolare si occupa di:

- Gestire le relazioni con l'esterno;
- Pianificare le *attività*$_G$: quali svolgere, data di inizio e fine, assegnazione delle priorità;
- Valutare i rischi delle scelte da effettuare;
- Controllare i progressi del progetto;
- Gestire le risorse umane;
- Approvazione della documentazione.

4.1.2.5 Amministratore

Questa figura professionale è incaricata del controllo e dell'amministrazione dell'ambiente di lavoro utilizzato dal gruppo ed è anche il punto di riferimento per quanto concerne le norme di progetto. Le sue mansioni principali sono:

- Affrontare e risolvere le problematiche associate alla gestione dei *processi*$_G$;
- Gestire versionamento della documentazione;
- Gestire la configurazione del prodotto;

- Redigere ed attuare le norme e le procedure per la gestione della qualità;
- Amministrare le infrastrutture e i servizi per i *processi*$_G$ di supporto.

4.1.2.6 Analista

Figura professionale con competenze avanzate riguardo l'*attività*$_G$ di *analisi dei requisiti*$_G$ ed il dominio applicativo del problema. Il suo ruolo è quello di identificare, documentare e comprendere a fondo le esigenze e le specifiche del progetto, traducendole in requisiti chiari e dettagliati. Si occupa di:

- Analizzare il contesto di riferimento, definire il problema in esame e stabilire gli obiettivi da raggiungere;
- Comprendere il problema e definire la complessità e i requisiti;
- Redigere il documento *Analisi dei Requisiti*;
- Studiare i bisogni espliciti ed impliciti.

4.1.2.7 Progettista

Il progettista è la figura di riferimento per quanto riguarda le scelte progettuali partendo dal lavoro dell'analista. Spetta al progettista assumere decisioni di natura tecnica e tecnologica, oltre a supervisionare il processo di sviluppo. Tuttavia, non è responsabile della manutenzione del prodotto.
In particolare si occupa di:

- Progettare l'*architettura*$_G$ del prodotto secondo specifiche tecniche dettagliate;
- Prendere decisioni per sviluppare soluzioni che soddisfino i criteri di affidabilità, efficienza, sostenibilità e conformità ai requisiti;
- Redige la *Specifica Architetturale* e la parte pragmatica del *Piano di Qualifica*;

4.1.2.8 Programmatore

Il programmatore è la figura professionale incaricata della scrittura del codice *software*$_G$.
Il suo compito primario è implementare il codice conformemente alle specifiche fornite dall'analista e all'*architettura*$_G$ definita dal progettista.
In particolare, il programmatore:

- Scrive codice manutenibile in conformità con le *Specifiche Tecniche*;
- Codifica le varie componenti dell'*architettura*$_G$ seguendo quanto ideato dai progettisti;

- Realizza gli strumenti per verificare e validare il codice;
- Redige il *Manuale Utente*.

4.1.2.9 Verificatore

La principale responsabilità del verificatore consiste nell'ispezionare il lavoro svolto da altri membri del team per assicurare la qualità e la conformità alle attese prefissate. Stabilisce se il lavoro è stato svolto correttamente sulla base delle proprie competenze tecniche, esperienza e conoscenza delle norme.
In particolare il verificatore si occupa di:

- Verificare che i prodotti siano conformi alle *Norme di Progetto*;
- Verificare la conformità dei prodotti ai requisiti funzionali e di qualità;
- Ricercare ed in caso segnalare eventuali errori;
- Redigere la sezione retrospettiva del *Piano di Qualifica*, descrivendo le verifiche e le prove effettuate durante il processo di sviluppo del prodotto.

4.1.2.10 Ticketing

GitHub è adottato come *sistema*$_G$ di tracciamento delle *issue*$_G$ (ITS), garantendo così una gestione agevole e trasparente delle *attività*$_G$ da svolgere.
L'amministratore ha la facoltà di creare e assegnare specifiche *issue*$_G$ sulla base delle *attività*$_G$ identificate dal responsabile, assicurando chiarezza sulle responsabilità di ciascun membro del team e stabilendo tempi definiti entro cui ciascuna *attività*$_G$ deve essere completata.
Inoltre, ogni membro del gruppo può monitorare i progressi compiuti nel periodo corrente, consultando lo stato di avanzamento delle varie *issue*$_G$ attraverso le Dashboard:

- DashBoard: per una panoramica dettagliata sullo stato delle *issue*$_G$;
- RoadMap: per una panoramica temporale dettagliata delle *issue*$_G$.

Procedura per la creazione delle issue

Le *issue*$_G$ vengono create dall'amministratore e devono essere specificati i seguenti attributi:

- **Titolo**: un titolo coinciso e descrittivo;
- **Descrizione**:
  - Descrizione testuale oppure "to-do" tramite bullet points;
  - Nell'ultima riga viene specificato il verificatore della *issue*$_G$ nel formato: "Verificatore: Mario Rossi".

- **Assegnatario**: incaricato/i allo svolgimento della *issue*$_G$;
- **Scadenza**: data entro la quale la *issue*$_G$ deve essere completata;
- **Labels**: tag per identificare la categoria della *issue*$_G$. (ex. Verbale, Documents, Develop, Bug, Feature).
  Inoltre per associare ad ogni *issue*$_G$ un Configuration Item vengono utilizzati i seguenti *label*$_G$:
  - **NdP:** Norme di progetto;
  - **PdQ:** Piano di Qualifica;
  - **PdP:** Piano di Progetto;
  - **AdR:** Analisi dei Requisiti;
  - **Poc:** Proof of concept;
  - **Gls:** Glossario.
- **Milestone**: *milestone*$_G$ associata alla *issue*$_G$;
- **Projects**: progetti a cui la *issue*$_G$ è associata.
  Se sono presenti *dashboard*$_G$ associate ad un progetto, le *issue*$_G$ correlate a tale progetto verranno visualizzate nella relativa/e *dashboard*$_G$ di progetto;
- **Development:** *branch*$_G$ e Pull Request associate alla *issue*$_G$.
  Quando una Pull Request viene accettata, la relativa *issue*$_G$ viene automaticamente chiusa ed eventualmente spostata nella sezione "Done" della Dashboard di progetto.

Ciclo di vita di una issue

Il ciclo di vita di una *issue*$_G$ è il seguente:

1. L'amministratore accede alla *repository*$_G$ GitHub, crea la *issue*$_G$ e la assegna ad un assegnatario, seguendo la convenzione descritta nel *paragrafo* precedente;
2. L'amministratore accede alla *dashboard*$_G$ di progetto e sposta la *issue*$_G$ dalla colonna "No Status" alla colonna "To Do";
3. L'assegnatario apre un *branch*$_G$ su GitHub seguendo la denominazione suggerita in "*Sincronizzazione e Branching*";
4. Quando la *issue*$_G$ viene presa in carico dall'assegnatario, questo accede alla DashBoard e sposta la *issue*$_G$ dalla colonna "To Do" alla colonna "In Progress";

5. Una volta che la *issue*$_G$ è considerata terminata, l'assegnatario apre una Pull Request su GitHub seguendo la convenzione descitta in dettaglio nella sezione "*Procedura per la creazione di Pull Request*".

6. All'interno della Dashboard GitHub la *issue*$_G$ deve essere spostata dalla colonna "In Progress" alla colonna "Da revisionare";

7. Il verificatore o i verificatori designati seguono le procedure esposte nella *sezione 3.2* per verificare le modifiche apportate al progetto;

8. Se la verifica ha esito positivo, la *issue*$_G$ viene trasferita dalla colonna "Da revisionare" alla colonna "Done" della Dashboard di GitHub.
   Nel caso in cui la *issue*$_G$ sia associata ad una Pull Request, una volta che quest'ultima viene accettata dal verificatore, la *issue*$_G$ viene automaticamente chiusa e spostata nella colonna "Done" della Dashboard di progetto.

4.1.2.11 Strumenti

- **GitHub:** *piattaforma*$_G$ utilizzata per il tracciamento e la gestione delle *issue*$_G$.

4.1.3 Coordinamento

4.1.3.1 Descrizione

Il coordinamento rappresenta l'*attività*$_G$ che sovraintende la gestione della comunicazione e la pianificazione degli incontri tra le diverse parti coinvolte in un progetto di ingegneria del *software*$_G$.
Questo comprende sia la gestione della comunicazione interna tra i membri del team del progetto, sia la comunicazione esterna con il *proponente*$_G$ e i committenti. Il coordinamento risulta essere cruciale per assicurare che il progetto proceda in modo efficiente e che tutte le parti coinvolte siano informate e partecipino attivamente in ogni fase del progetto.

4.1.3.2 Obiettivi

Il coordinamento in un progetto è fondamentale per gestire la comunicazione e pianificare gli incontri tra gli *stakeholder*$_G$.
L'obiettivo principale è garantire efficienza, evitando ritardi e confusioni, assicurando che tutte le parti in causa siano informate e coinvolte in ogni fase del progetto.
Inoltre, promuove la collaborazione e la coesione nel team, facilitando lo scambio di idee e la risoluzione dei problemi in modo collaborativo, creando un ambiente lavorativo positivo e produttivo.

Comunicazione

Il gruppo *ByteOps* mantiene comunicazioni attive, sia interne che esterne al team, le quali possono essere sincrone o asincrone, a seconda delle necessità.

4.1.3.3 Comunicazioni sincrone

- **Comunicazioni sincrone interne**
  Per le comunicazioni sincrone interne, il gruppo ByteOps, ha scelto di adottare *Discord*$_G$ in quanto permette di comunicare tramite chiamate vocali, videochiamate, messaggi di testo, media e file in chat private o come membri di un *"server Discord"*;

- **Comunicazioni sincrone esterne**
  Per le comunicazioni sincrone esterne,in accordo con l'azienda *proponente*$_G$ si è deciso di utilizzare Google Meet.

4.1.3.4 Comunicazioni asincrone

- **Comunicazioni asincrone interne**
  Le comunicazioni asincrone interne avvengono tramite l'applicazione *Telegram*$_G$ all'interno di un gruppo dedicato, il quale consente una comunicazione rapida tra tutti i membri del gruppo. Inoltre, tramite la stessa *piattaforma*$_G$, è possibile avere conversazioni dirette e private (chat) tra due membri;

- **Comunicazioni asincrone esterne**
  Per le comunicazioni asincrone esterne sono stati adottati due canali differenti:

  – **E-mail:** utilizzata per comunicare con il *committente*$_G$ e per coordinare gli incontri con l'azienda *proponente*$_G$;

  – **Element:** client di messaggistica istantanea gratuito ed open source che supporta conversazioni strutturate e crittografate. La sua flessibilità nell'adattarsi a varie esigenze di comunicazione, inclusa la possibilità di condividere file, immagini e altri documenti, ha reso la *piattaforma*$_G$ un'opzione versatile e completa per soddisfare le esigenze specifiche del nostro contesto lavorativo. Si fa uso della *piattaforma*$_G$ Element come canale di comunicazione con l'azienda *proponente*$_G$ per richiedere eventuali chiarimenti e informazioni di natura urgente, evitando l'attesa del meeting dedicato allo stato di avanzamento lavori (*SAL*$_G$), il quale potrebbe tenersi a diversi giorni di distanza.

Riunioni

In ogni riunione, qualunque ne sia la tipologia, verrà designato un segretario con l'incarico di prendere appunti durante il meeting e successivamente redigere un verbale completo, documentando gli argomenti trattati e i risultati emersi durante le discussioni.

4.1.3.5 Riunioni interne

Si è scelto di svolgere i meeting interni a cadenza settimanale, al fine di facilitare una comunicazione costante e coordinare il progresso delle *attività*$_G$.
Generalmente le riunioni sono programmate per ogni venerdi alle ore:

- **10:00** nel caso in cui non sia previsto un *SAL*$_G$ con l'azienda *proponente*$_G$ nello stesso giorno;
- **11:30** nel caso in cui sia previsto un *SAL*$_G$ con l'azienda *proponente*$_G$ nello stesso giorno.

Se qualche membro del gruppo non può partecipare alla riunione nella data e nell'orario stabiliti, si procede programmando un nuovo incontro, concordando data e ora tramite un sondaggio sul canale Telegram dedicato.

Ogni membro del gruppo ha la facoltà di richiedere una riunione supplementare se necessario. In questo caso, la data e l'orario saranno concordati sempre attraverso il canale Telegram dedicato, mediante la creazione di un sondaggio.

Le riunioni interne rivestono un ruolo cruciale nel monitorare il progresso delle mansioni assegnate, valutare i risultati conseguiti e affrontare i dubbi e le difficoltà che possono sorgere. Durante i meeting interni, i membri del team condividono gli aggiornamenti sulle proprie *attività*$_G$, identificano le problematiche riscontrate e discutono di opportunità di miglioramento nei *processi*$_G$ di lavoro. Questo ambiente aperto e collaborativo favorisce l'interazione, l'innovazione e la condivisione di nuove prospettive.

Per agevolare la comunicazione sincrona, il canale utilizzato per i meeting interni è *Discord*$_G$, ritenuto particolarmente efficace per tali scopi.

Relativamente ai meeting interni, sarà compito del responsabile:

- Stabilire preventivamente i principali temi da trattare durante la riunione, considerando la possibilità di aggiungerne di nuovi nel corso della riunione stessa;
- Guidare la discussione e raccogliere i pareri dei membri in maniera ordinata;
- Nominare un segretario per la riunione;
- Pianificare e proporre le nuove *attività*$_G$ da svolgere.

Verbali interni

Lo svolgimento di una riunione interna ha come obiettivo la retrospettiva del periodo precedente, la discussione dei punti stilati nell'ordine del giorno e la pianificazione delle nuove *attività*$_G$.
Alla conclusione di ciascuna riunione, l'amministratore apre un'issue nell'ITS di GitHub e assegna l'incarico di redigere il verbale interno al segretario della riunione. È compito quindi del segretario redigere il verbale, includendo tutte le informazioni rilevanti emerse durante la riunione. Le indicazioni dettagliate per la compilazione dei verbali interni sono disponibili nella *sezione 3.1.6.5*.

4.1.3.6 Riunioni esterne

Durante lo svolgimento del progetto, è essenziale organizzare vari incontri con i *Committenti* e/o il *Proponente* al fine di valutare lo stato di avanzamento del prodotto e chiarire eventuali dubbi o questioni.

La responsabilità di convocare tali incontri ricade sul responsabile, il quale è incaricato di pianificarli e agevolarne lo svolgimento in modo efficiente ed efficace.
Sarà compito del responsabile anche l'esposizione dei punti di discussione al *proponente*$_G$/*committente*$_G$, lasciando la parola ai membri del gruppo interessati quando necessario. Questo approccio assicura una comunicazione efficace tra le varie parti in causa, garantendo una gestione ottimale del tempo e una registrazione accurata delle informazioni rilevanti emerse durante gli incontri.

I membri del gruppo si impegnano a garantire la propria presenza in modo costante alle riunioni, facendo il possibile per riorganizzare eventuali altri impegni al fine di partecipare. Nel caso in cui gli obblighi inderogabili di un membro del gruppo rendessero impossibile la partecipazione, il responsabile assicurerà di informare tempestivamente il *proponente*$_G$ o i committenti, richiedendo la possibilità di rinviare la riunione ad una data successiva.

Riunioni con l'azienda proponente

In accordo con l'azienda *proponente*$_G$, si è stabilito di tenere incontri di *stato avanzamento lavori*$_G$ (*SAL*$_G$) con cadenza bisettimanale tramite Google Meet.
Durante tali incontri, si affrontano diversi aspetti, tra cui:

- Discussione delle *attività*$_G$ svolte nel periodo precedente, valutando l'aderenza a quanto concordato e identificando eventuali problematiche riscontrate;
- Pianificazione delle *attività*$_G$ per il prossimo periodo, definendo gli obiettivi e le azioni necessarie per il loro raggiungimento;
- Chiarezza e risoluzione di eventuali dubbi emersi nel corso delle *attività*$_G$ svolte.

Verbali esterni

Come nel caso delle riunioni interne, anche per le riunioni esterne verrà redatto un verbale con le stesse modalità descritte nella sezione relativa ai *Verbali Interni*.
Le linee guida per la redazione dei verbali esterni sono reperibili alla *sezione 3.1.6.5*.

4.1.3.7 Strumenti

- **Discord:** impiegato per la comunicazione sincrona e i meeting interni del team;
- **Element:** utilizzato per contattare l'azienda *proponente*$_G$ per richieste di chiarimenti, informazioni e per esporre dubbi;
- **Google Meet:** utilizzato per i *SAL*$_G$ e per i meeting con l'azienda *proponente*$_G$;
- **Telegram:** utilizzato per la comunicazione asincrona con il team;
- **GMail:** utilizzato come *servizio*$_G$ di posta elettronica.

4.1.3.8 Metriche

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

Table 8: Metriche relative alla gestione dei processi

4.2 Miglioramento

4.2.1 Introduzione

Secondo lo *standard*$_G$ ISO/IEC 12207:1995, il processo di miglioramento nel ciclo di vita del *software*$_G$ è finalizzato a stabilire, misurare, controllare e migliorare i *processi*$_G$ che lo compongono. L'*attività*$_G$ di miglioramento è composta da:

- **Analisi:** Identificare le aree di miglioramento dei *processi*$_G$;
- **Miglioramento:** Implementare le modifiche necessarie per migliorare i *processi*$_G$. di sviluppo del *software*$_G$;

4.2.2 Analisi

Questa operazione richiede di essere eseguita regolarmente e ad intervalli di tempo appropriati e costanti. L'analisi fornisce un ritorno sulla reale efficacia e correttezza dei *processi*$_G$ implementati, permettendo di identificare prontamente quelli che necessitano di miglioramenti.

Durante ogni riunione, il team dedica inizialmente del tempo per condurre una retrospettiva sulle *attività*$_G$ svolte nell'ultimo periodo. Questa pratica implica una riflessione approfondita su ciò che è stato realizzato, coinvolgendo tutti i membri nella identificazione delle aree di successo e di possibili miglioramenti.
L'obiettivo principale è formulare azioni correttive da implementare nel prossimo sprint, promuovendo così un costante feedback e un adattamento continuo per migliorare le prestazioni complessive del team nel corso del tempo.

4.2.3 Miglioramento

Il team implementa le azioni correttive stabilite durante la retrospettiva, successivamente valuta la loro efficacia e le sottopone nuovamente a esame durante la retrospettiva successiva.
L'esito di ogni azione correttiva sarà documentato nella sezione "Revisione del periodo precedente" di ogni verbale.

4.3 Formazione

4.3.1 Introduzione

L'obiettivo di questa iniziativa è stabilire *standard*$_G$ per il processo di apprendimento all'interno del team, assicurando la comprensione adeguata delle conoscenze necessarie per la realizzazione del progetto.

Si prevede che il processo di formazione del Team assicuri che ciascun membro acquisisca una competenza adeguata per utilizzare consapevolmente le tecnologie selezionate dal gruppo per la realizzazione del progetto.

4.3.2 Metodo di formazione

4.3.2.1 Individuale

Ciascun membro del team si impegnerà in un processo di autoformazione per adempiere alle *attività*$_G$ assegnate al proprio ruolo. Durante la rotazione dei ruoli, ogni membro del gruppo condurrà una riunione con il successivo occupante del suo attuale ruolo, trasmettendo le

conoscenze necessarie. Al contempo, terrà una riunione con chi ha precedentemente svolto il ruolo che esso assumerà, con l'obiettivo di apprendere le competenze richieste.

4.3.2.2 Di gruppo

Sono programmate sessioni formative, condotte dalla *proponente*$_G$, al fine di trasferire competenze relative alle tecnologie impiegate nel contesto del progetto. La partecipazione del team a tali riunioni è obbligatoria.

5 Standard per la qualità

Nel corso dell'analisi e della valutazione della qualità dei *processi*$_G$ e del *software*$_G$, adotteremo *standard*$_G$ internazionali ben definiti per garantire una valutazione rigorosa e conforme agli *standard*$_G$ globali. In particolare, l'utilizzo dello *standard*$_G$ ISO/IEC 9126 fornirà una solida struttura per valutare la qualità del *software*$_G$, concentrandosi su attributi quali la funzionalità, l'affidabilità, l'usabilità, l'efficienza, la manutenibilità e la portabilità. Questo *framework*$_G$ ci consentirà di misurare in modo accurato e completo la qualità del prodotto *software*$_G$.

Parallelamente, la suddivisione dei *processi*$_G$ in primari, di supporto e organizzativi sarà guidata dall'adozione dello *standard*$_G$ ISO/IEC 12207:1995. Infine, l'adozione dello *standard*$_G$ ISO/IEC 25010 ci fornirà un quadro completo per la definizione e la suddivisione delle metriche di qualità del *software*$_G$. L'utilizzo congiunto di questi *standard*$_G$ consentirà un approccio completo e strutturato alla valutazione della qualità dei *processi*$_G$ e del *software*$_G$, assicurando un'elevata coerenza, affidabilità e conformità agli *standard*$_G$ riconosciuti a livello internazionale.

5.1 Caratteristiche del Sistema, Standard ISO/IEC 25010

5.1.1 Funzionalità

- **Idoneità Funzionale:** la capacità del *software*$_G$ di fornire funzionalità che soddisfano i requisiti specificati;
- **Accuratezza:** la precisione con cui il *software*$_G$ esegue le sue funzioni;
- **Interoperabilità:** la capacità del *software*$_G$ di interagire con altri sistemi.

5.1.2 Affidabilità

- **Maturità:** la capacità del *software*$_G$ di evitare errori o malfunzionamenti;
- **Tolleranza agli Errori:** la capacità di mantenere un certo livello di prestazioni nonostante la presenza di errori;
- **Recuperabilità:** la capacità del *software*$_G$ di ripristinarsi dopo un errore.

5.1.3 Usabilità

- **Comprensibilità:** la facilità con cui gli utenti possono comprendere il *software*$_G$;
- **Apprendibilità:** il tempo e lo sforzo richiesti per apprendere a utilizzare il *software*$_G$;
- **Operabilità:** la facilità con cui gli utenti possono operare il *software*$_G$.

5.1.4 Efficienza

- **Tempo di Risposta:** il tempo impiegato dal *software*$_G$ per rispondere alle richieste dell'utente;
- **Utilizzo delle Risorse:** l'efficienza nell'uso delle risorse del *sistema*$_G$.

5.1.5 Manutenibilità

- **Analizzabilità:** la facilità con cui è possibile analizzare il codice per individuare errori o problemi;
- **Modificabilità:** la facilità con cui il *software*$_G$ può essere modificato;
- **Stabilità:** la capacità di evitare effetti indesiderati durante o dopo le modifiche.

5.1.6 Portabilità

- **Adattabilità:** la facilità con cui il *software*$_G$ può essere adattato a diversi ambienti;
- **Installabilità:** la facilità con cui il *software*$_G$ può essere installato;
- **Conformità:** il rispetto delle norme e degli *standard*$_G$ relativi alla portabilità.

5.2 Suddivisione secondo Standard ISO/IEC 12207:1995

5.2.1 Processi primari

Sono gli elementi centrali e fondamentali delle *attività*$_G$ di sviluppo del *software*$_G$. Questi *processi*$_G$ sono direttamente coinvolti nella produzione del prodotto *software*$_G$, dalla sua concezione alla sua realizzazione effettiva. Essi comprendono le *attività*$_G$ principali di acquisizione dei requisiti, progettazione, implementazione, *test*$_G$ e manutenzione del *software*$_G$.

5.2.2 Processi di supporto

Sono i *processi*$_G$ che forniscono supporto ai *processi*$_G$ primari durante tutto il ciclo di vita del *software*$_G$. Questi *processi*$_G$ comprendono *attività*$_G$ come la gestione della configurazione, la garanzia della qualità, la gestione dei documenti, la formazione, la pianificazione e il controllo dei progetti. Il loro obiettivo principale è garantire un ambiente efficiente e ben organizzato per il corretto svolgimento dei *processi*$_G$ primari.

5.2.3 Processi organizzativi

Rappresentano i *processi*$_G$ che definiscono la struttura e l'ambiente organizzativo all'interno del quale avvengono le *attività*$_G$ di sviluppo del *software*$_G$.

6 Metriche di qualità

Le metriche di qualità dei prodotti *software*$_G$ e dei *processi*$_G$ sono strumenti fondamentali per valutare e migliorare l'efficacia e l'efficienza nello sviluppo del *software*$_G$. Queste metriche forniscono indicatori oggettivi e misurabili che consentono di valutare la conformità agli *standard*$_G$, identificare aree di miglioramento e monitorare la salute complessiva del processo di sviluppo.

6.1 Metriche per la qualità di processo

- **Metrica M1PMS:**
  - **Nome:** Percentuale di Metriche Soddisfatte (PMS);
  - **Descrizione:** Misura che valuta quante metriche che sono state definite sono state effettivamente adottate o soddisfatte;
  - **Formula:** $\frac{metriche\ soddisfatte}{metriche\ totali} \times 100$
- **Metrica M2EAC:**
  - **Nome:** Estimated at Completion (EAC_G);
  - **Descrizione:** Misura il costo realizzativo stimato per terminare il progetto;
  - **Formula:** $EAC_G = \frac{BAC}{CPI}$
- **Metrica M3CPI:**
  - **Nome:** Cost Performance Index (CPI_G);
  - **Descrizione:** Misura il rapporto tra il valore del lavoro effettivamente svolto ed il costo reale del lavoro fino al periodo di riferimento;
  - **Formula:** $CPI_G = \frac{EV}{AC}$
- **Metrica M4BV:**
  - **Nome:** Budget Variance (BV);
  - **Descrizione:** Misura la differenza percentuale di budget tra quanto previsto nella pianificazione di un periodo e l'effettiva realizzazione;
  - **Formula:** $BV = EV - AC$
- **Metrica M5AC:**
  - **Nome:** Actual Cost (AC);
  - **Descrizione:** Misura i costi effettivamente sostenuti dall'inizio del progetto fino all'attualità;
  - **Formula:** Dato disponibile e aggiornato in "Piano di progetto" per ogni periodo.

- **Metrica M6SV:**
  - **Nome:** Schedule Variance (SV);
  - **Descrizione:** Indica in percentuale quanto si è in anticipo o in ritardo con le *attività*$_G$ pianificate;
  - **Formula:** $SV = EV - PV_G$
- **Metrica M7EV:**
  - **Nome:** Earned Value (EV);
  - **Descrizione:** Valore del lavoro effettivamente svolto fino a quel periodo;
  - **Formula:** $EV = \%lavoro\ svolto \times EAC_G$
- **Metrica M8PV:**
  - **Nome:** Planned Value (PV_G);
  - **Descrizione:** Stima la somma dei costi realizzativi delle *attività*$_G$ imminenti periodo per periodo;
  - **Formula:** $PV_G = \%lavoro\ svolto \times BAC$
- **Metrica M9ETC:**
  - **Nome:** Estimate to Complete (ETC_G);
  - **Descrizione:** Stima i costi realizzativi fino alla fine del progetto;
  - **Formula:** $ETC_G = EAC_G - AC$
- **Metrica M11RNP:**
  - **Nome:** Rischi Non Previsti (RNP);
  - **Descrizione:** Misura il numero di rischi non previsti nel corso del progetto.

- **Metrica M12VR:**
  - **Nome:** Variazione dei Requisiti (VR);
  - **Descrizione:** Misura la variazione nei requisiti dal momento della pianificazione;
  - **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 Superati (PCTS);
  - **Descrizione:** Percentuale di casi di $test_G$ superati;
  - **Formula:** $\frac{numero\ di\ casi\ di\ test\ superati}{numero\ totale\ di\ casi\ di\ test} \times 100$

- **Metrica M14PCTF:**
  - **Nome:** Percentuale di Casi di Test Falliti (PCTF);
  - **Descrizione:** Percentuale di casi di $test_G$ non superati;
  - **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 il numero di istruzioni che sono state eseguite almeno una volta;
  - **Formula:** $MSC_G = \frac{istruzioni\ eseguite}{istruzioni\ totali} \times 100$

- **Metrica M16BC:**
  - **Nome:** Branch Coverage (MBC);
  - **Descrizione:** Indice di quante diramazioni del codice vengono eseguite dai *test*$_G$. Un *branch*$_G$ è uno dei possibili percorsi di esecuzione che il codice può seguire dopo che un'istruzione decisionale viene valutata;
  - **Formula:** $MBC = \frac{flussi\ funzionali\ implementati\ e\ testati}{flussi\ condizionali\ riusciti\ e\ non} \times 100$

- **Metrica M17CNC:**
  - **Nome:** CoNdition Coverage (CNC);
  - **Descrizione:** Metrica di copertura del codice che indica la percentuale di condizioni logiche nel codice sorgente che sono state eseguite durante i *test*$_G$;
  - **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 che valuta quanto del lavoro svolto durante lo sviluppo corrisponda ai requisiti essenziali o obbligatori definiti in fase di *analisi dei requisiti*$_G$;
  - **Formula:** $\frac{requisiti\ obbligatori\ soddisfatti}{requisiti\ obbligatori\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.

- **Metrica M19PRDS:**
  - **Nome:** Percentuale di Requisiti Desiderati Soddisfatti (PRDS);
  - **Descrizione:** Metrica usata per valutare quanti di quei requisiti, che se integrati arricchirebbero l'esperienza dell'utente o fornirebbero vantaggi aggiuntivi non strettamente necessari, sono stati implementati o soddisfatti nel prodotto;
  - **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 per valutare quanti dei requisiti aggiuntivi, non essenziali o di bassa priorità, sono stati implementati o soddisfatti nel prodotto;
  - **Formula:** $\frac{requisiti\ opzionali\ soddisfatti}{requisiti\ opzionali\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.

- **Metrica M21IF:**
  - **Nome:** Implementazione delle Funzionalità (IF);
  - **Descrizione:** Misura qual è la quantità di funzionalità pianificate che sono state implementate;
  - **Formula:**$(1 - \frac{F_{NL}}{F_L}) \times 100$
  - **Caratteristica di qualità:** Funzionalità.

- **Metrica M22CO:**
  - **Nome:** Correttezza Ortografica (CO);
  - **Descrizione:** Rappresenta il numero di errori grammaticali ed ortografici all'interno di un documento;
  - **Caratteristica di qualità:** Affidabilità.

- **Metrica M23IG:**
  - **Nome:** Indice Gulpease (MIG);
  - **Descrizione:** Indice di leggibilità di un testo tarato sulla lingua italiana, che utilizza la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico;
  - **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:** Misura la percentuale di errori presenti nel prodotto rispetto al totale del 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 livello di accoppiamento tra le classi del *sistema*$_G$;
  - **Caratteristica di qualità:** Manutenibilità.

- **Metrica M26MCCM**
  - **Nome:** Complessità Ciclomatica per Metodo (*MCCM*$_G$);
  - **Descrizione:** Misura il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso del metodo;
  - **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:** Numero massimo di parametri per metodo;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M28APC:**
  - **Nome:** Attributi Per Classe (APC);
  - **Descrizione:** Misura il numero massimo di attributi per classe;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M29LCM:**
  - **Nome:** Linee di Codice per Metodo (LCM);
  - **Descrizione:** Limite massimo di linee di codice per metodo;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M30PG:**
  - **Nome:** Profondità delle Gerarchie (PG);
  - **Descrizione:** Metrica che misura il numero di livelli tra una classe base (super-classe) e le sue sotto-classi (classi derivate);
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M31TMR:**
  - **Nome:** Tempo Medio di Risposta (TMR);
  - **Descrizione:** Metrica che misura quanto è efficiente e reattivo un *sistema*$_G$ *software*$_G$;
  - **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 misura l'usabilità di un *sistema*$_G$ *software*$_G$;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M33TA:**
  - **Nome:** Tempo di Apprendimento (TA);
  - **Descrizione:** Misura il tempo massimo richiesto per apprendere l'utilizzo del prodotto;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M34VBS:**
  - **Nome:** Versioni dei Browser Supportate (VBS);
  - **Descrizione:** Metrica che misura la percentuale delle versioni di browser supportate rispetto al totale delle versioni disponibili;
  - **Caratteristica di qualità:** Portabilità.