Norme di Progetto

v0.3

7Last

Versioni

Ver.	Data	Autore	Verificatore	Descrizione
0.3	05/03/2024	Raul Seganfreddo		Aggiunta Introduzione
0.2	05/04/2024	Matteo Tiozzo		Modificato tabella versioni
0.1	29/03/2024	Raul Seganfreddo		Creazione struttura documento

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di descrivere le regole e le procedure che il gruppo adotterà per lo svolgimento del progetto. Lo scopo quindi è quello di definire il *Way of Working* del gruppo, in modo da garantire un lavoro efficiente e di qualità.
Il processo seguirà le linee guida descritte dallo standard ISO/IEC 12207:1995.

1.2 Scopo del progetto

Lo scopo del progetto è quello di realizzare una piattaforma di monitoraggio per una smart city, in grado di raccogliere e analizzare in tempo reale dati provenienti da diverse fonti, come sensori, dispositivi indossabili e macchine. La piattaforma avrà lo scopo di:

- Migliorare la qualità della vita dei cittadini: la piattaforma consentirà alle autorità locali di prendere decisioni informate e tempestive sulla gestione delle risorse e sull'implementazione di servizi, basandosi su dati reali e aggiornati.
- Coinvolgere i cittadini: i dati monitorati saranno resi accessibili al pubblico attraverso portali online e applicazioni mobili, permettendo ai cittadini di essere informati sullo stato della loro città e di partecipare attivamente alla sua gestione.
- Gestire il big data: la piattaforma sarà in grado di gestire grandi volumi di dati provenienti da diverse fonti, aggregandoli, normalizzandoli e analizzandoli per estrarre informazioni significative.

La piattaforma si baserà su tecnologie di data streaming processing per l'analisi in tempo reale dei dati e su una piattaforma OLAP per la loro archiviazione e visualizzazione. La parte "IoT" del progetto sarà simulata attraverso tool di generazione di dati realistici.
In sintesi, il progetto mira a creare una piattaforma che sia:

- Efficiente: in grado di raccogliere e analizzare grandi volumi di dati in tempo reale.
- Efficace: in grado di fornire informazioni utili per la gestione della città e il miglioramento della qualità della vita dei cittadini.
- Accessibile: in grado di rendere i dati disponibili al pubblico in modo chiaro e comprensibile.

Il progetto si pone come obiettivo di contribuire allo sviluppo di città più intelligenti e sostenibili, in cui la tecnologia viene utilizzata per migliorare il benessere dei cittadini.

1.3 Glossario

Al fine di evitare ambiguità e di facilitare la comprensione del documento, si allega il *Glossario* contenente la definizione dei termini tecnici e degli acronimi utilizzati.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato d'appalto C6 SyncCity: Smart city monitoring platform
  `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6p.pdf`
- *ISO/IEC 12207:1995*:
  `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf`

1.4.2 Riferimenti informativi

- Glossario: **TODO: inserire link**.

2 Processi primari

2.1 Fornitura

2.1.1 Introduzione

Il processo di fornitura ha lo scopo di stabilire un accordo contrattuale tra il fornitore e il cliente, in cui vengono definiti i servizi che il fornitore si impegna a fornire e le condizioni di fornitura. Il processo di fornitura comprende le seguenti attività:

2.1.2 Attività

- **Preparazione della proposta**
- **Contrattazione**
- **Pianificazione**
- **Esecuzione**
- **Revisione**
- **Consegna**

2.1.3 Comunicazioni con l'azienda proponente

L'azienda proponente SyncLab mette a disposizione l'indirizzo di posta elettronica e il suo canale Discord per la comunicazione tramite messaggi e Google Meet per la comunzicazione attraverso incontri telematici. Gli incontri telematici hanno una cadenza regolare di due settimane, con possibili incontri aggiuntivi richiesti dal gruppo in caso di necessità, come ad esempio chiarimenti riguardo al capitolato o alle tecnologie utilizzate. Per ogni colloquio avvenuto con l'azienda proponente verrà fornito un verbale esterno che riporterà i vari argomenti discussi durante il colloquio. I verbali saranno disponibili all'interno del *repository* `https://github.com/7Last/docs`

2.1.4 Documentazione fornita

Di seguito saranno elencati i documenti che il gruppo *7last* consegnerà all'azienda *SyncLab* e ai committenti *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*.

2.1.4.1 Valutazione dei capitolati

Il presente documento offre una valutazione approfondita dei capitolati d'appalto presentati in data 2023-10-17. Per ciascun progetto, vengono esaminate le richieste del proponente, le possibili soluzioni e le eventuali criticità. La valutazione si articola nelle seguenti sezioni:

- **Descrizione**: viene elencato il nome del progetto, l'azienda proponente, i committenti e l'obbiettivo del progetto;
- **Dominio Applicativo**: viene descritto il contesto del progetto;
- **Dominio Tecnologico**: vengono descritte le tecnologie utilizzate per lo sviluppo del progetto;
- **Aspetti Positivi**;
- **Aspetti Negativi**;

2.1.4.2 Analisi dei requisiti

L'Analisi dei Requisiti è un documento completo che delinea i casi d'uso, i requisiti e le funzionalità necessarie per il prodotto software. Il suo scopo principale è chiarire qualsiasi incertezza o ambiguità che potrebbe sorgere dopo la lettura del capitolato. Questo documento include:

- Una **descrizione dettagliata del prodotto**;
- Un **elenco dei casi d'uso**: riporta tutti gli scenari possibili in cui il sistema software potrebbe essere utilizzato dagli utenti finali, descrivendo le azioni che gli utenti compiono nel sistema in modo da identificare requisiti non ovvi inizialmente;
- Un **elenco dei requisiti**: specifica tutti i vincoli richiesti dal proponente o dedotti in base all'analisi dei casi d'uso associati ad essi.

2.1.4.3 Piano di progetto

Il Piano di Progetto è un documento che si propone di delineare la pianificazione e la gestione delle attività necessarie per portare a termine il progetto. Esso comprende le seguenti informazioni:

- **Analisi dei Rischi**: identificazione delle potenziali problematiche che potrebbero emergere durante lo sviluppo e che potrebbero rallentare o ostacolare il progresso del progetto. Il gruppo si impegna a fornire soluzioni per tali problemi il prima possibile. I rischi sono classificati in due categorie principali: rischi organizzativi e rischi tecnologici;
- **Modello di sviluppo**: descrizione dell'approccio metodologico e strutturato adottato dal gruppo per lo sviluppo del prodotto;
- **Pianificazione**: delineamento dei periodi temporali, con gli eventi e le attività correlate, all'interno di un calendario. Per ogni periodo, saranno specificate le attribuzioni dei ruoli e una stima dell'impegno richiesto da ciascun membro del gruppo per svolgere le rispettive attività;
- **Preventivo**: stima della durata di ciascun periodo, indicando il tempo necessario per completare tutte le attività pianificate;
- **Consultivo**: analisi del lavoro effettivamente svolto rispetto a quanto preventivato, al fine di ottenere uno stato di avanzamento del progetto al termine di ciascun periodo.

2.1.4.4 Piano di qualifica

Il Piano di Qualifica è un documento che dettaglia le responsabilità e le attività del Verificatore all'interno del progetto. Queste attività sono cruciali per garantire la qualità del prodotto software in fase di sviluppo. Il Piano di Qualifica funge da guida essenziale per la gestione del processo di sviluppo, poiché assicura che il prodotto finale soddisfi le specifiche richieste e le aspettative del committente, monitorando il suo progresso rispetto agli obiettivi stabiliti. Ogni membro del team coinvolto nel progetto farà riferimento a questo documento per garantire il raggiungimento della qualità desiderata. Il Piano di Qualifica è strutturato in diverse sezioni, tra cui:

- **Qualità di processo**: definisce i parametri e le metriche per garantire processi di alta qualità;
- **Qualità del prodotto**: stabilisce i parametri e le metriche per assicurare un prodotto finale di alta qualità;
- **Test**: descrive i test necessari per verificare il soddisfacimento dei requisiti nel prodotto;

- **Valutazioni per il miglioramento**: riporta le attività di verifica svolte e le problematiche riscontrate durante lo sviluppo del software, con l'obiettivo di identificare aree di miglioramento.

2.1.4.5 Glossario

Il *Glossario* è una raccolta di termini presenti nei documenti, accompagnati dalle relative definizioni, specialmente quando il loro significato potrebbe non essere immediatamente chiaro. Serve a prevenire eventuali ambiguità e facilitare la comunicazione tra i membri del gruppo.

2.1.4.6 Lettera di presentazione

La Lettera di Presentazione è il documento attraverso il quale il gruppo *7Last* manifesta l'intenzione di partecipare alla fase di revisione del prodotto software. Questo documento elenca la documentazione disponibile per i committenti e il proponente, nonché i termini concordati per la consegna del prodotto finito.

2.1.5 Strumenti

Di seguito sono descritti gli strumenti software impiegati nel processo di fornitura.

2.1.5.1 Discord

Il gruppo utilizza Discord come piattaforma per le riunioni interne e come un metodo informale per contattare l'azienda proponente tramite messaggistica e videochat.

2.1.5.2 Latex

LaTeX è un sistema di preparazione di documenti utilizzato principalmente per la creazione di documenti tecnici e scientifici.

2.1.5.3 Git

Git è un software per il controllo di versione.

2.1.6 GitHub

GitHub è un servizio di hosting per progetti software.

2.2 Sviluppo

2.2.1 Introduzione

L'ISO/IEC 12207:1995 fornisce le linee guida per il processo di sviluppo, che comprende attività cruciali come analisi, progettazione, codifica, integrazione, testing, installazione e accettazione. È essenziale eseguire tali attività in stretta conformità alle linee guida e ai requisiti stabiliti 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 rappresenta un'attività cruciale nello sviluppo del software poiché fornisce le fondamenta per il design, l'implementazione e i test del sistema. Secondo lo standard ISO/IEC 12207:1995, l'obiettivo dell'analisi dei requisiti è comprendere e definire in modo completo le necessità del cliente e del sistema. Questa attività richiede di rispondere a domande fondamentali come "Qual è il contesto?", "Quali sono i requisiti essenziali del cliente?", e implica una comprensione approfondita del contesto e la definizione chiara degli obiettivi, dei vincoli e dei requisiti sia tecnici che funzionali.

2.2.2.2 Obiettivi

- Collaborare con la proponente per definire gli obiettivi del prodotto al fine di soddisfare le aspettative, includendo l'identificazione, la documentazione e la validazione dei requisiti funzionali e non funzionali;
- Promuovere una comprensione condivisa tra tutte le parti interessate;
- Consentire una stima accurata delle tempistiche e dei costi del progetto;
- Fornire ai progettisti requisiti chiari e facilmente comprensibili;
- Agevolare l'attività di verifica e di test fornendo indicazioni pratiche di riferimento.

2.2.2.3 Documentazione

È responsabilità degli analisti condurre l'analisi dei requisiti, redigendo un documento omonimo che deve comprendere i seguenti elementi:

- **Introduzione**: presentazione e scopo del documento stesso;
- **Descrizione**: analisi approfondita del prodotto, includendo:
  - Obbiettivi del prodotto;
  - Funzionalità del prodotto;
  - Caratteristiche utente;
  - Tecnologie impiegate.
- Casi d'uso: descrizione delle funzionalità offerte dal sistema dal punto di vista dell'utente, includendo:
  - utenti esterni al sistema;
  - Elenco dei casi d'uso, comprensivo di:
    * Descrizioni dei casi d'uso in formato testuale;
    * Diagrammid dei casi d'uso.
  - Eventuali diagrammi di attività per facilitare la comprensione dei processi relativi alle funzionalità.
- **Requisiti**:
  - Requisiti funzionali;
  - Requisiti qualitativi;
  - Requisiti di vincolo.

2.2.2.4 Casi d'uso

I casi d'uso forniscono una dettagliata descrizione delle funzionalità del sistema dal punto di vista degli utenti, delineando come il sistema risponde a specifiche azioni o scenari. Essenzialmente, i casi d'uso sono strumenti utilizzati nell'analisi dei requisiti per catturare e illustrare chiaramente e comprensibilmente come gli utenti interagiranno con il software e quali saranno i risultati di tali interazioni.

Ogni caso d'uso testuale deve includere:

1. **Identificativo**:

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

   ad esempio **TODO: inserire esempio**
   con

   - **Numero caso d'uso**: identificativo numerico del caso d'uso;
   - **Numero sotto caso d'uso**: identificativo numerico del sotto caso d'uso (presente solo se si tratta di un sotto caso d'uso);
   - **Titolo**: breve e chiaro titolo del caso d'uso.

2. **Attore principale**: entità esterna che interagisce attivamente con il sistema per soddisfare una propria necessità.

3. **Attore secondario**: eventualmente, un'entità esterna che non interagisce attivamente con il sistema, ma all'interno del caso d'uso consente al sistema di soddisfare la necessità dell'attore principale.

4. **Descrizione**: una breve descrizione della funzionalità, se necessaria.

5. **Scenario principale**: una sequenza di eventi che si verificano quando un attore interagisce con il sistema per raggiungere l'obiettivo del caso d'uso (postcondizioni).

6. **Estensioni**: eventuali scenari alternativi che si verificano in seguito a una o più specifiche condizioni, portando il flusso del caso d'uso a non raggiungere le postcondizioni.

7. **Precondizioni**: lo stato in cui deve trovarsi il sistema affinché la funzionalità sia disponibile per l'attore.

8. **Postcondizioni**: lo stato in cui si trova il sistema dopo l'esecuzione dello scenario principale.

9. **User story associata**: una breve descrizione di una funzionalità del software, scritta dal punto di vista dell'utente, che fornisce contesto, obiettivi e valore.

   - L'user story viene scritta nella forma: "Come [utente] desidero poter [funzionalità] per [valore aggiunto]".

2.2.2.5 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono strumenti grafici che consentono di rappresentare in modo chiaro e intuitivo le funzionalità fornite dal sistema dal punto di vista dell'utente. Inoltre, permettono di individuare e comprendere rapidamente le relazioni e le interazioni tra i diversi casi d'uso, offrendo una visione generale delle funzionalità del sistema. Questi diagrammi si concentrano sulla descrizione delle funzionalità del sistema dal punto di vista degli utenti, senza entrare nei dettagli implementativi. La loro principale finalità è quella di evidenziare le interazioni esterne al sistema, fornendo una visione focalizzata sulle funzionalità e sull'interazione dell'utente con il sistema stesso. Un diagramma dei casi d'uso fornisce una panoramica visuale delle principali interazioni tra gli attori e il sistema, agevolando la comprensione dei requisiti funzionali del sistema e la comunicazione tra le parti interessate del progetto. Di seguito sono elencati i principali componenti di un diagramma dei casi d'uso: **TODO: inserire componenti diagramma casi d'uso**

2.2.2.6 Requisiti

I requisiti di un software sono dettagliate specifiche documentate che delineano le funzionalità, le prestazioni, i vincoli e altri aspetti critici che il software deve soddisfare. Questi requisiti sono fondamentali per guidare lo sviluppo, il testing e la valutazione del prodotto, garantendo che risponda alle esigenze degli utenti e agli obiettivi del progetto. Essi comprendono sia i **requisiti funzionali**, che descrivono le funzionalità necessarie, sia i **requisiti non funzionali**, che definiscono criteri di prestazione, qualità, sicurezza e vincoli del sistema.
Una definizione precisa dei requisiti è essenziale: devono essere chiari e rispondere completamente alle aspettative del cliente o del proponente.

Ogni requisito è costituito da:

1. **identificativo** nel formato

   R[Abbreviazione Tipologia Requisito][Codice]

   con:

   - **Abbreviazione Tipologia Requisito**: indica la tipologia del requisito, che può essere:

  - **RF**: requisito funzionale;
  - **RQ**: requisito di qualità;
  - **RV**: requisito di vincolo.
- **Codice**: numero progressivo che identifica univocamente il requisito.

2. **Importanza**: indica il grado di importanza del requisito, che può essere:
   - **Obbligatorio**: requisito essenziale per il funzionamento del sistema;
   - **Desiderabile**: requisito che apporta valore aggiunto al sistema, ma non essenziale;
   - **Opzionale**: requisito che può essere implementato in un secondo momento.
3. **Descrizione**: descrizione chiara e dettagliata che fornisce una spiegazione del comportamento o della funzionalità richiesta.
4. **Fonte**: indica la fonte da cui è stato identificato il requisito, che può essere:
   - **Capitolato**: requisito identificato direttamente dal capitolato d'appalto;
   - **Verbale Interno**: requisito identificato durante un incontro interno;
   - **Verbale Esterno**: requisito identificato durante un incontro con il proponente.
5. **Casi d'uso**: elenco dei casi d'uso che soddisfano il requisito.

2.2.2.7 Metriche

Nell'analisi dei requisiti, le metriche sono strumenti essenziali per valutare, misurare e gestire diversi aspetti dei requisiti di un sistema o di un progetto. Grazie a queste metriche, è possibile garantire che i requisiti siano esaustivi, precisi, coerenti e comprensibili.

Metrica	Abbreviazione	Riferimento
1M-EV	Earned Value	1M-EV
2M-PV	Planned Value	2M-PV
3M-AC	Actual Cost	3M-AC

Tabella 1: Metriche per l'analisi dei requisiti

TODO: da cambiare la tabella

2.2.2.8 Strumenti

StarUML è un'applicazione software impiegata dal team per creare i diagrammi dei casi d'uso.

2.2.3 Progettazione

2.2.3.1 Descrizione

Il principale obiettivo dell'attività di progettazione è individuare la soluzione implementativa ottimale che soddisfi pienamente le esigenze di tutti gli stakeholder, considerando i requisiti e le risorse disponibili. La progettazione si pone la domanda fondamentale: "Qual è il modo migliore per realizzare ciò di cui c'è bisogno?". È essenziale definire l'architettura del prodotto prima di iniziare la fase di codifica, adottando un approccio basato sulla correttezza per costruzione anziché sulla correzione successiva. Tale approccio consente di gestire efficacemente 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 garantire che i requisiti siano soddisfatti attraverso un sistema di qualità definito dall'architettura del prodotto. Ciò comporta:

- Individuare componenti modulari che rispettino i requisiti, con specifiche chiare e coerenti, e svilupparle utilizzando risorse sostenibili e costi contenuti;
- Organizzare le componenti in modo che siano facilmente comprensibili e manutenibili, garantendo una struttura coesa e ben organizzata;
- Definire un'architettura che supporti l'evoluzione del prodotto, consentendo l'aggiunta di nuove funzionalità e la correzione di eventuali errori;

Inizialmente, il team di progettazione eseguirà un'analisi approfondita per selezionare con cura le tecnologie più adeguate, valutandone attentamente i vantaggi, i limiti e le eventuali problematiche. Una volta individuate le tecnologie appropriate, si procederà allo sviluppo di un'architettura di alto livello per comprendere e delineare la struttura generale del prodotto, che fungerà da base iniziale per la realizzazione del *Proof of Concept* (PoC). Questa architettura fornirà una visione panoramica del sistema, identificando i principali componenti, i flussi di dati e le interazioni tra di essi, ponendo particolare attenzione alla flessibilità del sistema per eventuali modifiche future.

Successivamente, si darà il via allo sviluppo del PoC, una parte cruciale della *Technology Baseline*, per valutare le decisioni prese riguardo all'architettura e alle tecnologie adottate, e per verificare la loro congruenza con gli obiettivi e le specifiche del progetto. Dopo lo sviluppo e un'attenta analisi del PoC, si procederà con ulteriori iterazioni, apportando miglioramenti, aggiustamenti e integrazioni fino a raggiungere un design completo. Questo design sarà fondamentale per lo sviluppo del *Minimum Viable Product* (MVP), che rappresenterà una versione essenziale e funzionale del prodotto e sarà parte integrante della *Product Baseline*.

2.2.3.3 Documentazione

Specifica tecnica
Il documento fornisce una visione dettagliata del design definitivo del prodotto e offre istruzioni chiare agli sviluppatori per implementare correttamente la soluzione software, seguendo i requisiti e le specifiche indicate. Questo aiuta a semplificare il processo di sviluppo del software, riducendo la complessità e le ambiguità, e assicurando che il prodotto finale sia in linea con le aspettative del cliente e funzioni in modo ottimale. Tra gli elementi chiave inclusi in questo documento vi sono:

- **Tecnologie utilizzate**: elenco delle tecnologie, dei framework e degli strumenti impiegati per lo sviluppo del prodotto;
- **Architettura logica**: descrizione dettagliata della struttura logica del sistema, con particolare attenzione ai componenti principali, ai flussi di dati e alle interazioni tra di essi;
- **Architettura di deployment**: rappresentazione grafica dell'architettura del sistema, con indicazioni sulle risorse hardware e software necessarie per il corretto funzionamento del prodotto;
- **Design pattern**: descrizione dei design pattern utilizzati per risolvere problemi comuni e ricorrenti durante lo sviluppo del software;
- **Vincoli e linee guida**: specifiche restrizioni e regole da seguire durante lo sviluppo del prodotto, per garantire coerenza e uniformità nel codice.
- **Procedure di testing e validazione**: indicazioni sulle procedure e gli strumenti da utilizzare per verificare e validare il prodotto, garantendo che soddisfi i requisiti e le aspettative del cliente.

- **Requisiti tecnici**: elenco dettagliato dei requisiti tecnici che il prodotto deve soddisfare, con indicazioni sulle funzionalità, le prestazioni e le caratteristiche richieste.

2.2.3.4 Qualità dell'architettura

- **Sufficienza**: l'architettura deve soddisfare tutti i requisiti funzionali e non funzionali del sistema, garantendo che tutte le funzionalità richieste siano implementate correttamente e che il sistema funzioni in modo ottimale.
- **Comprensibilità**: l'architettura deve essere chiara, ben strutturata e facilmente comprensibile, in modo che gli sviluppatori possano capire facilmente come il sistema è organizzato e come funziona;
- **Modularità**: l'architettura deve essere modulare, con componenti ben definiti e indipendenti, in modo che possano essere facilmente riutilizzati e sostituiti senza influenzare il resto del sistema;
- **Robustezza**: l'architettura deve essere robusta e resistente agli errori, in modo che il sistema possa gestire eventuali problemi o malfunzionamenti senza interrompere il funzionamento del sistema;
- **Flessibilità**: l'architettura deve essere flessibile e adattabile, in modo che il sistema possa essere facilmente modificato e ampliato per soddisfare nuove esigenze e requisiti;
- **Efficienza**: l'architettura deve essere efficiente e ottimizzata, in modo che il sistema possa funzionare in modo rapido ed efficiente, senza sprechi di risorse;
- **Riusabilità**: l'architettura deve essere progettata per favorire la riutilizzabilità dei componenti, in modo che possano essere facilmente utilizzati in altri contesti e progetti;
- **Affidabilità**: l'architettura deve essere affidabile e sicura, in modo che il sistema possa garantire la corretta esecuzione delle funzionalità e la protezione dei dati e delle informazioni;
- **Disponibilità**: l'architettura deve garantire la disponibilità del sistema, in modo che possa essere sempre accessibile e operativo per gli utenti;

- **Safety**: l'architettura deve garantire la sicurezza del sistema, in modo che possa proteggere i dati e le informazioni sensibili in seguito a malfunzionamenti;
- **Security**: l'architettura deve garantire la sicurezza del sistema, in modo che possa proteggere i dati e le informazioni sensibili da accessi non autorizzati.
- **semplicità**: l'architettura deve essere semplice e intuitiva, in modo che possa essere facilmente compresa e utilizzata dagli sviluppatori e dagli utenti.
- **Coesione**: l'architettura deve essere coesa, con componenti ben definiti e correlati tra loro, in modo che possano lavorare insieme in modo efficace e armonioso;
- **Incapsulazione**: l'architettura deve essere incapsulata, con componenti ben definiti e indipendenti, in modo che possano essere facilmente gestiti e mantenuti senza influenzare il resto del sistema;
- **Basso accoppiamento**: l'architettura deve avere un basso accoppiamento tra i componenti, in modo che possano essere facilmente sostituiti e modificati senza influenzare il resto del sistema;

2.2.3.5 Diagrammi UML

Vantaggi:

- **Chiarezza nella comunicazione**: i diagrammi UML forniscono una rappresentazione visuale delle informazioni, facilitando la comprensione e la comunicazione tra gli stakeholder;
- **Standardizzazione**: UML è uno standard riconosciuto a livello internazionale, che consente di creare diagrammi coerenti e uniformi, garantendo una maggiore coerenza e comprensibilità;
- **Analisi e progettazione visiva**: i diagrammi UML consentono di analizzare e progettare il sistema in modo visuale, facilitando la comprensione e l'identificazione di problemi e soluzioni;
- **Modellazione e simulazione**: UML consente di modellare e simulare il sistema in modo visuale, facilitando la valutazione delle prestazioni e delle funzionalità del sistema;

- **Manutenzione facilitata**: i diagrammi UML semplificano la manutenzione del sistema, consentendo di identificare e risolvere facilmente problemi e bug;
- **Riduzione degli errori di progettazione**: UML aiuta a ridurre gli errori di progettazione, consentendo di identificare e correggere i problemi in modo rapido ed efficace;
- **Documentazione supportata**: i diagrammi UML forniscono una documentazione visuale del sistema, che facilita la comprensione e la consultazione delle informazioni.

A supporto della progettazione, il team utilizzerà i seguenti **diagrammi delle classi**.

Diagrammi delle classi

Ogni diagramma delle classi rappresenta le proprietà e le relazioni tra le varie componenti di un sistema, offrendo una visione chiara e dettagliata della struttura del sistema. Le classi sono rappresentate da rettangoli suddivisi in tre sezioni:

1. **Nome della classe**: indica il nome della classe;
2. **Attributi**: elenco degli attributi della classe, con il relativo tipo di dato, seguendo il formato:

Visibilità Nome: Tipo [Molteplicità] = Valore di default

- **Visibilità**: indica il livello di accesso agli attributi, che può essere:
  - **+**: pubblico;
  - **-**: privato;
  - **#**: protetto;
  - **~**: package.
- **Nome**: nome dell'attributo. Deve essere rappresentativo, chiaro e deve seguire la notazione *nomeAttributo: tipo*;
  Se l'attributo è costante, il nome deve essere scritto in maiuscolo (es. *PIGRECO: double*);
- **Molteplicità**: nel caso di una sequenza di elementi come liste o array, indica il numero di elementi presenti, se questa non fosse conosciuta si utilizza il simbolo ***** (es *tipoAttributo*[*]);

- **default**: valore di default dell'attributo.

3. **Metodi**: descrivono il comportamento della classe, seguendo il formato:

Visibilità Nome(parametri): Tipo di ritorno

- **Visibilità**: indica il livello di accesso ai metodi, che può essere:
  - **+**: pubblico;
  - **-**: privato;
  - **#**: protetto;
  - **~**: package.
- **Nome**: nome del metodo. Deve essere rappresentativo, chiaro e deve seguire la notazione *nomeMetodo*(*parametri*)*: tipoRitorno;*
- **Parametri**: elenco dei parametri del metodo, separati tramite virgola. Ogni parametro deve seguire la notazione *nomeParametro: tipo;*
- **Tipo di ritorno**: indica il tipo di dato restituito dal metodo.

Convenzioni sui metodi

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

Relazioni tra le classi TODO: inserire relazioni tra le classi

2.2.3.6 Design pattern

I design pattern rappresentano solide soluzioni a problemi ricorrenti di progettazione in determinati contesti, offrendo un approccio riutilizzabile che assicura la qualità e una rapida implementazione. Si adottano quando una soluzione ha dimostrato efficacia in un contesto specifico. Le guide dettagliate sull'applicazione dei pattern delineano il loro utilizzo ottimale, accompagnate da rappresentazioni grafiche, spiegazioni testuali

della logica e descrizioni della loro utilità nell'architettura complessiva. Questa documentazione è essenziale per favorire una comprensione approfondita dell'integrazione dei design pattern nell'architettura generale e per prevenire errori di progettazione.

2.2.3.7 Test

Nel processo di sviluppo, il testing è cruciale per garantire la qualità del prodotto finale. Durante questa fase, vengono stabiliti i requisiti di testing, definiti i casi di test e i criteri di accettazione, che servono da strumenti per valutare il software.
L'obiettivo principale è individuare e risolvere eventuali problemi o errori nel software prima del rilascio del prodotto finale, assicurando che soddisfi le specifiche e le aspettative del cliente. I progettisti hanno il pieno controllo su questa attività, incluso il definire i test da eseguire. Nella sezione 3.2.4 (**TODO: controllare se è il numero giusto**) sono fornite descrizioni dettagliate delle varie tipologie di test e della terminologia associata, offrendo ulteriore chiarezza su questa fase critica del processo di sviluppo del software.

2.2.3.8 Metriche

TODO: inserire metriche

2.2.3.9 Strumenti

StarUML è un'applicazione software impiegata dal team per creare i diagrammi dei casi d'uso.

2.2.4 Codifica

2.2.4.1 Descrizione

Nel processo di sviluppo del software, la codifica è responsabilità del programmatore ed è il momento cruciale in cui le funzionalità richieste prendono vita. Durante questa fase, le idee e i concetti delineati dai progettisti vengono tradotti in codice, creando istruzioni e procedure eseguibili dai calcolatori.
È essenziale che i programmatori rispettino attentamente le linee guida e le norme stabilite per assicurare che il codice sia conforme alle specifiche e rifletta accuratamente le visioni iniziali dei progettisti.

2.2.4.2 Obiettivi

La fase di codifica mira a sviluppare un prodotto software che soddisfi le richieste del cliente e sia conforme agli accordi concordati. Il rispetto rigoroso delle norme assicura la creazione di codice di elevata qualità, semplificando la manutenzione, l'espansione e la verifica del software. Questo contribuisce costantemente al miglioramento complessivo della sua qualità.

2.2.4.3 Norme di codifica

Le seguenti norme sono state formalizzate in questa maniera:

- **Nomi significativi**: i nomi delle variabili, delle costanti, delle classi e dei metodi devono essere significativi e rappresentativi della loro funzione, in modo da facilitare la comprensione del codice;
- **Indentazione e formattazione consistente**: il codice deve essere correttamente indentato e formattato, con l'uso di spazi e tabulazioni coerenti, per garantire una corretta leggibilità e comprensione;
- **Lunghezza dei metodi**: i metodi devono essere brevi e concisi, con un numero limitato di righe di codice, per garantire una maggiore chiarezza e facilità di manutenzione. I metodi dovrebbero essere lunghi quanto basta per svolgere una singola funzione. Questo favorisce:
  - **Chiarezza**: i metodi brevi sono più facili da comprendere e da seguire;
  - **manutenibilità**: i metodi brevi sono più facili da modificare e da aggiornare;
  - **Comprensibilità**: i metodi brevi sono più facili da leggere e da interpretare;
  - **Testabilità**: i metodi brevi sono più facili da testare e da validare.
  - **Lunghezza del codice**: i file di codice sorgente devono essere brevi e concisi, con un numero limitato di righe di codice, per garantire una maggiore chiarezza e facilità di manutenzione.
  - **Commenti**: il codice deve essere corredato da commenti chiari e significativi, che spiegano il funzionamento e lo scopo delle varie parti del codice, per facilitare la comprensione e la manutenzione.
  - **Conformità ai principi SOLID**: il codice deve rispettare i principi SOLID, che promuovono la scrittura di codice pulito, modulare e manutenibile.

2.2.4.4 Strumenti

Visual Studio Code è l'editor di codice utilizzato dal team per scrivere e modificare il codice sorgente.

2.2.4.5 Metriche

TODO: inserire metriche

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

La redazione dei file Docker è inclusa nel processo di sviluppo del software. Rispettare le regole e le migliori pratiche di codifica per i file Docker è essenziale per assicurare la creazione, la gestione e la distribuzione efficiente dei container.

- **Chiarezza e Coerenza**: i file Docker devono essere chiari e coerenti, con una struttura ben definita e una formattazione uniforme, per garantire una corretta organizzazione e facilitare la comprensione;
- **Versionamento**: i file Docker devono includere informazioni dettagliate sulla versione del software e dei componenti utilizzati, per garantire la compatibilità e la coerenza tra i diversi ambienti;
- **Sicurezza**: i file Docker devono essere sicuri e protetti da eventuali vulnerabilità, per garantire la protezione dei dati e delle informazioni sensibili, utilizzando immagini ufficiali e aggiornate, evitando di eseguire comandi con privilegi elevati quando possibile;
- **Efficienza**: i file Docker devono essere efficienti e ottimizzati, con un utilizzo corretto delle risorse e una gestione accurata dei container, per garantire prestazioni elevate e tempi di risposta rapidi;
- **Gestione delle variabili d'ambiente**: i file Docker devono includere variabili d'ambiente ben definite e gestite correttamente, per garantire la configurabilità e la flessibilità del sistema;

- **Logging e monitoraggio**: bisogna configurare i *container* per registrare i log e monitorare le prestazioni, per garantire la visibilità e la tracciabilità delle attività del sistema;
- **Layering**: ridurre il numero di layer e mantenere il più possibile le dipendenze in comune tra i vari layer, per garantire una maggiore efficienza e una migliore gestione delle risorse.
- **Riduzione delle dimensioni delle immagini**: ridurre le dimensioni delle immagini Docker, eliminando i file e le dipendenze non necessarie, per garantire una maggiore efficienza e una migliore gestione delle risorse.
- **Documentazione**: i file Docker devono essere corredati da una documentazione dettagliata e aggiornata, che spieghi il funzionamento e la configurazione del sistema, per facilitare la comprensione e la manutenzione.
- **Testing**: i file Docker devono essere testati e validati accuratamente, per garantire che il sistema funzioni correttamente e soddisfi i requisiti e le aspettative del cliente.

2.2.5.2 Strumenti

- **Docker**: Docker è una piattaforma open source che semplifica la creazione, la distribuzione e la gestione di container, fornendo un ambiente isolato e sicuro per eseguire applicazioni e servizi.
- **Visual Studio Code**: Visual Studio Code è un editor di codice leggero e versatile, che offre funzionalità avanzate per lo sviluppo del software, tra cui il supporto per Docker e la creazione di file Docker.

3 Processi di supporto

3.1 Documentazione

3.1.1 Introduzione

Il processo di documentazione è una componente fondamentale nella realizzazione e nel rilascio di un prodotto *software*, poichè fornisce informazioni utili alle parti coinvolte e tiene traccia di tutte le attività relative al ciclo di vita del software, comprese scelte e *norme* adottate dal gruppo durante lo svolgimento del progetto. In particolare, la documentazione è utile per:

- Permettere una comprensione profonda del prodotto e delle sue funzionalità;
- Tracciare un confine tra disciplina e creatività;
- Garantire uno standard di qualità all'interno dei *processi* produttivi.

Lo scopo della sezione è:

- Fornire una raccolta esasutiva di regole che i membri del gruppo devono seguire per agevolare la stesura della documentazione;
- Definire delle procedure ripetibili per uniformare la redazione, la verifica e l'approvazione dei documenti;
- Creare template per ogni tipologia di documento così da garantire omogeneità e coerenza.

3.1.2 Documentation as Code

TODO adozione di questo approccio?

3.1.3 Tipografia e sorgente documenti

Per la redazione dei documenti abbiamo deciso di utilizzare il linguaggio di markup LaTeX, in quanto semplifica la creazione e la manutenzione dei documenti, liberando i redattori dall'onere della visualizzazione grafica e garantendo coerenza nella documentazione del progetto. Inoltre, per favorire una migliore collaborazione tra i diversi autori, abbiamo scelto di scomporre ogni documento in più file, ciascuno per una specifica sezione, in modo da permettere a più persone di lavorare sulle singole sezioni o

sottosezioni. Il risultato finale sarà ottenuto tramite l'assemblaggio di tutti i file sorgenti in un file principale, attraverso l'uso del comando *inputSezione.tex* o, nel caso delle sottosezioni, *inputSottosezione.tex*. Solo nel caso di documenti di piccole dimensioni, come i verbali, si potrà optare per la scrittura di un unico file.

3.1.4 Ciclo di vita

Il ciclo di vita di un documento è composto dalle seguenti fasi:

1. Pianificazione della stesura e suddivisione in sezioni: tramite confronto con il gruppo, le sezioni del documento vengono stabilite e assegnate ai Redattori. Essi sono responsabili della stesura delle proprie sezioni in conformità con le Norme di Progetto.
2. Stesura del contenuto e creazione della bozza iniziale: i Redattori realizzano il documento redigendone il contenuto e creano una prima bozza che viene utilizzata come punto di partenza per la discussione e la revisione.
3. Controllo dei contenuti: dopo la stesura effettiva del documento, i Redattori verificano che e il contenuto delle proprie sezioni sia conforme alle norme definite e non contenga errori di compilazione.
4. Revisione: quando la redazione del documento è conclusa, questo viene revisionato dai Verificatori incaricati.
5. Approvazione e rilascio: nell'ultima fase il documento viene approvato da un Responsabile e rilasciato in versione finale.

3.1.5 Procedure correlate alla redazione di documenti

3.1.5.1 I redattori

Il redattore è colui che si occupa di scrivere e curare il contenuto di un documento o di una sua sezione in modo chiaro, accurato e comprensibile. Nel farlo deve seguire lo stesso approccio impiegato nella codifica del *software*, adottando il *workflow* noto come *feature branch*. **Caso redazione nuovo documento/sezione o modifica dei precedenti già verificati** In queste situazioni il redattore dovrà creare un nuovo *branch* Git in locale e posizionarsi su di esso con i seguenti comandi:

- *git checkout main*

- *git checkout -b nomeBranch*

In particolare l'identificativo del branch deve essere 'parlante', ossia descrittivo e significativo così da consentire una compresione immediata del documento o della sezione che si sta redigendo. Dunque il redattore deve adottare le specifiche convenzioni per la nomenclatura dei branch. Una volta terminata la redazione del documento o della sezione assegnata, è necessario rendere disponibile il branch nella *repository* remota seguento la seguente procedura:

1. Eseguire il *push* delle modifiche nel *branch*:
   - *git add .*
   - *git commit -m "Descrizione delle modifiche apportate"*
   - *git push origin nomeBranch*
2. Se riscontriamo problemi nel punto 1: *git pull origin nomeBranch*
3. Risolviamo i conflitti e ripetiamo il punto 1.

Caso modifica documento in fase di redazione Per continuare la redazione di un documento o di una sezione già in fase di stesura, sono necessari i seguenti comandi

- *git pull*
- *git checkout nomeBranch*

Completamento redazione documento Dopo aver completato la redazione del documento o della sezione, il redattore deve procedere nel seguente modo:

1. Spostare l'*issue* relativa all'*attività* assegnata nella colonna "Review" della DashBoard del progetto, così da comunicare il completamento dell'incarico ai Verificatori.
2. Aggiornare la tabella contenente il versionamento del documento, inserendo le informazioni richieste e incrementando la versione.
3. Creare una Pull Request:
   (a) Accedere alla Repository GitHub, spostarsi nella sezione "Pull Request" e cliccare su "New Pull Request";

(b) Selezionare come *branch* di destinazione "main" e come *branch* sorgente il ramo creato appositamente per la redazione del documento/sezione;

(c) Cliccare su "Create Pull Request";

(d) Dare un titolo significativo e, se necessario, una descrizione alla Pull Request, selezionare i Verificatori e cliccare su "Create Pull Request".

3.1.5.2 I verificatori

Il ruolo e la procedura dei Verificatori sono descritti in dettaglio al paragrafo 3.2.2.

3.1.5.3 Il responsabile

Per quanto riguarda la redazione dei documenti, il Responsabile ha il dovere di:

- Identificare i documenti da redigere;
- Assegnare le task a Redattori e Verificatori;
- Stabilire la scadenza per il completamento delle attività;
- Approvare o richiedere eventuali modifiche ai documenti.

3.1.5.4 L'amministratore

L'amministratore è responsabile della gestione delle *attività* richieste dal Responsabile all'interno dell'ITS.

3.1.6 Struttura del documento

Tutta la documentazione prodotta segue uno schema strutturale ben definito e uniforme.

3.1.6.1 Prima pagina

Nella prima pagina di ogni documento è presente un'intestazione contenente le seguenti informazioni:

- Nome del documento;
- Versione del documento;

- Logo del gruppo;
- Nome del gruppo.

3.1.6.2 Registro delle modifiche

La seconda pagina è dedicata al registro delle modifiche in formato tabellare e permette di tenere traccia delle modifiche apportate al documento. La tabella riporta i seguenti dati:

- Versione del documento;
- Data di rilascio;
- Nome dell'autore;
- Nome del Verificatore;
- Descrizione della modifica.

3.1.6.3 Indice

Ogni documento contiene un indice delle sezioni e delle sottosezioni presenti al suo interno, in modo da facilitare la consultazione e la navigazione.

3.1.6.4 Intestazione

Ogni pagina del documento, ad eccezione della prima, contiene un'intestazione che riporta il nome del documento, la versione e il logo del team.

3.1.6.5 Verbali: struttura generale

I verbali costituiscono un report dettagliato dei meeting, con lo scopo di tenere traccia degli argomenti trattati, delle decisioni adottate e le azioni da intraprendere. Essi si suddividono in esterni o interni, a seconda che il meeting sia con persone esterne al gruppo o con i soli membri del team. La struttura in ogni caso è la medesima e prevede le seguenti sezioni:

- **Informazioni sulla riunione**:

  - Sede del meeting;
  - Orario di inizio e fine;
  - Partecipanti del gruppo;
  - Partecipanti esterni.

- **Corpo del documento**
  - **Revisione del periodo precedente**: Analisi dello stato delle *attività* e dell'approccio lavorativo, si discute di eventuali problemi riscontrati ma anche degli aspetti positivi in modo da incrementare e migliorare il way of working.
  - **Ordine del giorno**: Elenco di ciò che verrà discusso durante la riunione.
  - **Sintesi dell'incontro**: Breve riassunto delle discussioni e dei temi affrontati surante l'incontro.
  - **Decisioni prese**: Sezione che elenca in formato testuale le decisioni prese durante il meeting. Alcune di queste potrebbero risultare anche in "Attività individuate".
  - **Attività individuate**: Illustrazione dettagliata delle attività assegnate ai diversi membri del gruppo in forma tabellare, sono presenti le seguenti informazioni:
    * Nome della task;
    * ID dell'*issue* su GitHub;
    * Assegnatari.

- **Ultima pagina**: Solo nel caso di verbale esterno, è presente una sezione dedicata alla data e alla firma delle terze parti coinvolte.

3.1.7 Norme tipografiche

Nomi assegnati ai file Il nome dei documenti deve essere omogogeneo alla tipologia di appartenenza, deve essere in minuscolo e contenere un riferimento alla versione del documento. In particolare la nominazione dei file deve seguire la convenzione:

- **Verbali**: verbale_esterno/interno_AA_MM_DD_vX.Y;
- **Norme di Progetto**: norme_di_progetto_vX.Y;
- **Analisi dei Requisiti**: analisi_dei_requisiti_vX.Y;

- **Piano di Progetto**: piano_di_progetto_vX.Y;
- **Glossario**: glossario_vX.Y.

Stile del testo

- **Grassetto**:
  - Titoli di sezione;
  - Termini importanti;
  - Parole seguite da descrzione o elenchi puntati.
- **Corsivo**:
  - Nome del gruppo e dell'azienda proponente;
  - Termini presenti nel glossario;
  - Riferimenti a documenti esterni.
- **Maiuscolo**:
  - Acronimi;
  - Iniziali dei nomi;
  - Iniziali dei ruoli svolti dai membri del gruppo.

Regole sintattiche:

- Negli elenchi ogni voce deve terminare con ";", ad eccezione dell'ultima che prevede ".";
- I numeri razionali si scrivono utilizzando la virgola come separatore tra parte intera e parte decimale;
- Le date devono seguire lo standard internazionale ISO 8601, ossia YYYY-MM-DD.

3.1.8 Abbreviazioni

Segue un elenco delle abbreviazioni più comuni utilizzate nei documenti:

Abbreviazione	Scrittura Estesa
RTB	Requirements and Technology Baseline
PB	Product Baseline
CA	Customer Acceptance
ITS	Issue Tracking System
CI	Configuration Item
SAL	Stato Avanzamento Lavori

Tabella 2: Spiegazione delle abbreviazioni utilizzate nei documenti.

3.1.9 Strumenti

Gli strumenti utilizzati per la redazione dei documenti sono:

- **LaTeX**: utilizzato per la stesura dei documenti;
- **GitHub**: utilizzato per la gestione del versionamento e per la condivisione dei documenti;
- **Visual Studio Code**: utilizzato come editor di testo per la scrittura dei documenti attraverso l'estensione LaTeX Workshop.

3.2 Verifica

3.2.1 Introduzione

Il processo di verifica è fondamentale durante tutto il ciclo di vita del *software*, a partire dall'iniziale fase di progettazione fino alla sucessiva manutenzione. La verifica ha lo scopo di garantire che ciascuna *attività* sia corretta ed efficiente, identificando un processo di controllo per ogni prodotto realizzato. In particolare, ci si preoccupa che gli output del software (documentazione, codice sorgente, test...) siano conformi alle aspettative e ai requisiti specificati. Nel farlo è fondamentale applicare tecniche e analisi di test seguendo procedure definite e adottando criteri affidabili. Le attività di verifica sono svolte dai Verificatori, i quali sono responsabili di analizzare i prodotti e valutare la loro aderenza agli standard stabiliti. Il fulcro di questo processo è il *Piano di Qualifica*, un documento dettagliato che traccia il percorso della verifica. Questo fornisce linee guida per una valutazione accurata della qualità, delineando chiaramente gli obiettivi da raggiungere e i criteri di accettazione da rispettare.

3.2.2 Verifica dei documenti

Nell'ambito della documentazione, la verifica è un'attività cruciale per garantire la correttezza e l'accuratezza dei contenuti. Essa si suddivide in:

- **Revisione della correttezza tecnica**: assicura che tutte le informazioni siano corrette e coerenti con le norme stabilite;
- **Conformità alle norme**: verifica che il documento segua le linee guida e gli *standard* stabiliti per la formattazione, la struttura e lo stile;
- **Revisione ortografica e grammaticale**: controlla che il testo sia privo di errori ortografici, grammaticali e di punteggiatura.
- **Chiarezza e comprensibilità**: valuta la leggibilità del documento, verificando che il contenuto sia chiaro, comprensibile e privo di ambiguità;
- **Coerenza**: verifica che il documento sia omogeneo e coerente, sia internamente che con i documenti correlati.

3.2.3 Analisi

L'analisi è un processo che si occupa di valutare la qualità degli ogetti statici (documenti e codice sorgente) e dinamici (test ed esecuzione del software).

3.2.3.1 Analisi statica

L'analisi statica è un'*attività* di controllo che prescinde dall'esecuzione del prodotto e si basa su una revisione manuale o automatica del codice e della documentazione. Essa è fondamentale per verificare la presenza di proprietà desiderate e la conformità ai vincoli e per garantire che non siano presenti errori o difetti. L'analisi statica prevede due metodi di lettura:

3.2.3.2 Walkthrough

Questa tecnica prevede una lettura integrale e approfondita del prodotto, con l'obiettivo di individuare errori e difetti. Dunque lo scopo della verifica non è specifica per un determinato tipo di errore, ma generale. Inoltre il walkthrough è un approccio collaborativo che coinvolge il Verificatore e l'autore del prodotto, in particolare esso si svolge in quattro fasi:

1. **Pianificazione**: il Verificatore e l'autore si confrontano per individuare le proprietà e i vincoli che il prodotto deve soddisfare;
2. **Lettura**: il Verificatore esamina il prodotto, annotando errori e verificando la conformità ai vincoli;
3. **Discussione**: il Verificatore e l'autore discutono degli errori riscontrati e valutano le possibili soluzioni;
4. **Correzione**: l'autore apporta le modifiche concordate.

Nel caso di prodotti particolarmente complessi o di grandi dimensioni, il walkthrough può risultare dispendioso in termini di risorse, per questo motivo è più probabile adottarne l'impiego nelle fasi iniziali del progetto.

3.2.3.3 Inspection

Al contrario del walkthrough, l'inspection prevede una conoscenza preventiva degli elementi da verificare, i quali vengono organizzati in liste di controllo specifiche (*checklist*). Di conseguenza questo approccio risulta più rapido ed efficiente nel contesto di documenti o codice complessi e strutturati poichè consente di identificare tempestivamente e risolvere potenziali problematiche.

3.2.3.4 Analisi dinamica

L'analisi statica è un'*attività* di controllo che richiede l'esecuzione effettiva del codice con lo scopo di individuare discordanze tra i risultati ottenuti e il comportamento atteso del software. Il *test* costituisce la principale tecnica di analisi dinamica, rappresentato da esecuzioni del codice in un dominio di casi definito in precedenza dal Verificatore. Quest'ultimo è composto da tutti i possibili casi e dati di input che possono far emergere difetti o eventuali problemi di funzionamento e garantire la qualità del prodotto finale. Per assicurare l'*efficacia* di un test è necessario che esso sia ripetibile e decidibile, ossia che produca risultati coerenti e che possa essere eseguito più volte senza che i risultati siano influenzati da fattori esterni. Un altro aspetto importante è l'automazione del processo, realizzabile tramite l'uso di strumenti specifici (driver, stub, logger) che consentono di eseguire i test in modo automatico e di monitorare i risultati ottenuti.

3.2.4 Testing

Lo scopo del testing è quello di individuare errori e difetti nella componente soggetta a *test*, garantendo che il prodotto soddisfi i requisiti specificati e produca i risultati attesi. Per ogni *test* è necessario definire i seguenti aspetti:

- **Ambiente**: il sistema hardware e *software* all'interno del quale viene eseguito il *test*;
- **Stato iniziale**: i parametri iniziali del sistema prima dell'esecuzione del *test*;
- **Input**: i dati di input necessari per l'esecuzione del *test*;
- **Output**: i risultati attesi in relazione ad un determinato input;
- **Commenti**: eventuali note aggiuntive.

3.2.4.1 Test di unità

I *test* di unità sono finalizzati alla verifica di componenti *software* atomiche, ossia singole unità di codice, come classi, metodi o funzioni. Essi sono implementati principalmente durante la progettazione e devono essere eseguiti per primi, in quanto verificano il corretto funzionamento prima dell'integrazione con altre unità. Questi *test* sono eseguiti in modo isolato e indipendente dal resto del *sistema*, al seguente scopo è consentito l'utilizzo di mock e stub per simulare il comportamento di componenti non ancora sviluppate. In base al tipo di controllo che si vuole efettuare possiamo distinguere due connotazioni differenti:

- **Test funzionali**: verificano che l'unità testata produca i risultati attesi in base ai dati di input;
- **Test strutturali**: verificano la copertura di tutti i possibili cammini di esecuzione del codice.

3.2.4.2 Test di integrazione

Questi *test* vengono pianificati durante la fase di progettazione architetturale, successivamente ai *test* di unità. Essi verificano la corretta integrazione tra le diverse unità *software* precedentemente testate per garantire che lavorino sinergicamente secondo le specifiche del progetto. Inoltre, è possibile annullare le modifiche apportate in

modo da ripristinare uno stato sicuro nel caso si verifichino errori durante l'esecuzione di questo processo. Distinguiamo due approcci di integrazione:

- **Top-down**: l'*integrazione* avviene partendo dalle componenti di *sistema* che hanno più dipendenze e maggiore rilevanza esterna, garantendo la disponibilità immediata delle funzionalità di alto livello. Questo approccio prevedere l'utilizzo di molti oggetti simulati;
- **Bottom-up**: l'*integrazione* avviene partendo dalle componenti di *sistema* che hanno meno dipendenze e maggiore valore interno, ovvero quelle meno visibili all'utente. Questo comporta una fase di test più tardiva delle funzionalità utente.

3.2.4.3 Test di sistema

I *test* di sistema sono finalizzati alla verifica del corretto funzionamento dell'intero *sistema*. Ovvero si assicurano che tutte le componenti siano integrate correttamente e che tutti i requisiti *software* siano presenti e funzionanti. Al termine di questa fase l'applicazione esegue le funzioni previste in modo accurato e affidabile. I *test* di sistema sono pianificati successivamente ai test di integrazione.

3.2.4.4 Test di regressione

I *test* di regressione devono essere eseguiti ogni qualvolta vengono apportate delle modifiche al codice. Essi, infatti, hanno l'obiettivo di garantire che queste modifiche non abbiano introdotto nuovi difetti o compromettano le funzionalità precedentemente testate, evitando così il verificarsi di regressioni. Questi controlli prevedono la ripetizione mirata di *test* di unità, d'integrazione e di sistema preservando la stabilità del *sistema*.

3.2.4.5 Test di accettazione

I *test* di accettazione rappresentano un processo fondamentale prima del rilascio del prodotto finale. Essi verificano che tutti le aspettative degli utenti e i requisiti richiesti dal committente siano pienamente soddisfatti. Per questo motivo devono essere svolti necessariamente in presenza del committente.

3.2.4.6 Sequenza delle fasi di test

La sequenza ordinata delle fasi di *test* 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

3.2.4.8 Stato dei test

3.3 Validazione

3.3.1 Introduzione

La validazione è un processo che si occupa di verificare che il prodotto *software* sia in linea con i requisiti e con le aspettative del cliente. Di conseguenza, è fondamentale l'interazione diretta con il *committente* e il *proponente*, al fine di ottenere un feedback immediato e garantire un chiaro allineamento tra ciò che è stato prodotto e le aspettative degli utenti finali. Dunque lo scopo finale è avere un prodotto pronto per il rilascio, determinando la conclusione del ciclo di vita del *software*.

3.3.2 Procedura di validazione

In questo processo copre un ruolo fondamentale il *test* di accettazione che mira a garantire la validazione del prodotto. Infatti i diversi *test* elencati nella sezione 3.2.4 (*Testing*) costituiscono un input per la validazione. Essi dovranno verificare:

- Il soddisfacimento dei casi d'uso;
- La conformità del prodotto ai requisiti obbligatori;
- Il soddisfacimento di altri requisiti concordati con il *committente*.

3.4 Gestione della configurazione

3.4.1 Introduzione

La gestione della configurazione è un processo attuato durante tutto il ciclo di vita del *software*, infatti viene applicata a tutte le categorie di "artefatti" coinvolti. Essa si occupa di tracciare e controllare le modifiche della documentazione e del codice prodotto, detti Configuration Item (CI). Così facendo le modifiche apportate saranno accessibili in qualsiasi momento, garantendo la possibilità di verificare le motivazioni alla base dei cambiamenti effettuati e anche il ripristino di versioni precedenti.

3.4.2 Versionamento

La convenzione di versionamento adottata è nel formato X.Y dove:

- **X**: rappresenta il completamento in vista di una delle fasi del progetto e dunque viene incrementato al raggiungimento di *RTB*, *PB* ed eventuale CA.
- **Y**: rappresenta una versione intermedia e viene incrementata ad ogni modifica significativa del documento.

3.4.3 Repository

Il team utilizza due *repository*:

- Documentazione: contenente la documentazione prodotta;
- Codice: contenente il codice del progetto.

3.4.3.1 Struttura repository

Il *repository* dedicato alla documentazione è organizzato nel seguente modo:

- **Candidatura**:
  - **Verbali esterni**: contenente i verbali delle riunioni con le *proponenti*;
  - **Verbali interni**: contenente i verbali delle riunioni svolte all'interno del team;
  - **Lettera di presentazione**;
  - **Preventivo costi e assunzione impegni**;
  - **Valutazione dei capitolati**.

- **RTB**:
  - **Verbali**: contenente tutti i verbali prodotti durante il periodo di *RTB*, distinti tra esternie e interni;
  - **Analisi dei Requisiti**;
  - **Piano di Progetto**;
  - **Piano di Qualifica**;
  - **Glossario**;
  - **Norme di Progetto**.
- **PB**:

3.4.4 Sincronizzazione e branching

3.4.4.1 Documentazione

L'approccio adottato per la redazione della documentazione segue il *workflow* noto come *feature branch*. Ossia ogni attività è identificata da una specifica *issue* di GitHub (ClickUp?) e, prima dello svolgimento di ciascuna di esse, il componente interessato crea una diramazione del *branch* "develop". Tale metodologia permette di lavorare in modo isolato e parallelo nei rispettivi "workspace", massimizzando il lavoro ed evitando sovrascritture indesiderate. **Convenzioni per la nomenclatura dei branch relativi alle attività di redazione o modifica di documenti**

- Il nome del *branch* deve riportare il nome del documento che si vuole redarre o modificare;
- Nel caso dei verbali, il nome deve presentare anche la data della riunione: *verbale_interno_yy_mm_dd* (es. verbale_interno_24_03_05);
- Nel caso di redazione o modifica di una singola sezione di un documento, il nome del *branch* deve avere il formato: *nomeDocumento_nomeSezione* (es. norme_di_progetto_introduzione);

3.4.4.2 Sviluppo

Il team utilizza lo stile di flusso di lavoro Gitflow. **Flusso di lavoro Gitflow**

1. **Branch develop**: è il punto di avvio per nuove attività, si crea a partire dal *branch* "main";
2. **Branch release**: gestisce la preparazione del *software* per un rilascio e ammette solo modifiche minori e correzione di bug, si crea a partire dal *branch* "develop";
3. **Branch feature**: si occupa dello sviluppo di nuove funzionalità;
4. **Merge di feature in develop**: una volta completata un'attività, il *branch* feature viene unito a develop;
5. **Merge di release in develop e main**: dopo il completamento del *branch* realese, esso viene unito sia a develop che a main;
6. **Branch hotfix**: si occupa della correzione di bug critici riscontrati nella fase di produzione;
7. **Merge di hotfix in develop e main**: dopo il completamento del *branch* hotfix, esso viene unito sia a develop che a main per garantire coerenza tra le versioni.

3.4.4.3 Pull Request

Quando un'*attività* è completata, il componente che l'ha svolta crea una Pull Request per integrare le modifiche nel ramo principale. Il Verificatore ha il compito di verificare la correttezza del lavoro svolto e approvare la Pull Request. **Procedura per la creazione di una Pull Request**

1. Accedere alla pagina del *repository* e spostarsi sulla sezione "Pull Request";
2. Cliccare su "New Pull Request";
3. Selezionare il *branch* di partenza e quello di destinazione;
4. Cliccare su "Create Pull Request";
5. Assegnare un titolo alla Pull Request;
6. Aggiungere una descrizione delle modifiche apportate;
7. Assegnare un Verificatore;
8. Per convenzione, l'assegnatario è colui che richiede la Pull Request;

9. Selezionare le *label*;
10. Selezionare il progetto;
11. Selezionare la *milestone*;
12. Cliccare su "Create Pull Request";
13. Nel caso di conflitti seguire la procedura per la risoluzione proposta da *GitHub*.

3.4.5 Controllo di configurazione

3.4.5.1 Change Request

Per lo svolgimento di questo processo in modo ordinato e strutturato seguiamo lo *standard* ISO/IEC 12207:1995 che prevede le seguenti *attività*:

1. **Identificazione e registrazione** Ciascuna change request è identificata, registrata e documentata. L'identificazione avviene tramite la creazione di un'*issue* con allegata l'etichetta "Change request". Inoltre vengono riportate informazioni come la natura della modifica richiesta, la priorità e l'impatto sul progetto.
2. **Valutazione e analisi** La change request viene valutata e analizzata per determinare la fattibilità e l'impatto sul *sistema*. Vengono analizzati costi e benefici conseguenti alla modifica.
3. **Approvazione o rifiuto** La change request viene approvata o respinta in base a criteri come budget, tempo e priorità.
4. **Pianificazione delle modifiche** Se la change request viene approvata, viene pianificata la sua implementazione e l'integrazione nel ciclo di sviluppo del *software*.
5. **Implementazione** Le modifiche vengono effettivamente implementate, è fondamentale tenere traccia di ciò che viene fatto così da consentire una corretta documentazione e, se necessario, il *rollback* allo stato precedente.
6. **Verifica e validazione** Le modifiche vengono verificate e validate per garantire che non abbiano introdotto nuovi difetti e che siano conformi agli obiettivi prefissati.
7. **Documentazione** Tutte le fasi del progetto descritto vengono documentate in modo accurato per garantire trasparenza e tracciabilità.

8. **Comunicazione agli stakeholder** Il confronto con gli *stakeholder* deve essere mantenuto durante tutto l'arco del processo, in modo da mantenere trasparenza e fiducia.

3.4.6 Contabilità dello Stato di Configurazione

La contabilità dello stato di configurazione (o Configuration Status Accounting) è un'*attività* che si occupa di tenere traccia e montorare tutte le configurazioni di un *sistema software* durante tutto il ciclo di vita. Nello specifico esso mantiene la trasparenza e la tracciabilità delle modifiche relative ai Configuration Item (CI), mantenendo un registro accurato di tutte le *attività*. **Registrazione delle configurazioni**: registrazioni di informazioni dettagliate su ogni CI:

- **Documentazione**: nella prima pagina di ciascuno di essi troviamo le informazioni riguardanti la configurazione;
- **Sviluppo**: le informazioni relative alla configurazione sono inserite come prime righe di ciascun file sotto forma di commento.

Stato e cambiamenti: tracciamento dello stato attuale di ogni CI e delle modifiche effettuate, ossia versione attuale, revisioni e baseline.

- **Registro delle modifiche**: per tenere traccia dello stato di ciascun elemento di configurazione si utilizza il registro delle modifiche integrato in ognuno di essi.
- **Branching e DashBoard**: dal momento che ciascuna *issue* è associata ad un CI tramite *label*, è possibile verificare facilmente se ci sono *attività* correlate in corso nella colonna "In progress" della Dashboard di progetto.

Supporto per la gestione delle change request: registrazione e documentazione di tutte le modifiche effettuate ai CI in risposta alle richieste di modifica. Per la gestione di queste richieste si utilizza l'Issue Tracking System (ITS) di GitHub (ClickUp?) creando una *issue* con la label "Change request".

3.4.7 Release management and delivery

Secondo lo *standard* ISO/IEC 12207:1995, questo processo si occupa di tutto ciò che riguarda il rilascio e la distribuzione del *software*. Più precisamente, la Release Management and Delivery gestisce la release, la distribuzione e la documentazione correlata al prodotto *software* pronto per l'uso operativo.

Nel nostro caso, la pianificazione della release avviene in concomitanza con le baseline stabilite per il progetto didattico, ossia *RTB* (Requirements and Technology Baseline), *PB* (Product Baseline) ed eventualmente CA (Customer Acceptance). Affinchè questo processo termini con successo, risulta fondamentale che esso sia preceduto da un'attenta fase di verifica e validazione del prodotto.

3.4.7.1 Procedura per la creazione di una release

1. Accedere alla pagina del *repository*;
2. Spostarsi sulla sezione "Releases";
3. Cliccare su "Create a new release";
4. Cliccare su "Choose a tag" e selezionare il tag che identifica la versione da rilasciare. Nel caso non fosse presente, è possibile crearlo cliccando su "Create a new tag";
5. Selezionare come *branch* di destinazione il *branch* main;
6. Scrivere una descrizione della release;
7. Cliccare su "Publish release".

Conclusa la procedura, sarà possibile visionare la release nella sezione "Releases" del *repository*.

3.5 Joint review

3.5.1 Introduzione

3.5.2 Implementazione del processo

3.5.2.1 Revisioni periodiche

3.5.2.2 SAL

3.5.2.3 Revisioni ad hoc

3.5.2.4 Risorse per le revisioni

3.5.2.5 Elementi da concordare

3.5.2.6 Documenti e distribuzione dei risultati

3.5.3 Project management reviews

3.5.3.1 Introduzione

3.5.3.2 Stato del progetto

3.5.4 Revisioni tecniche

3.5.5 Strumenti

3.6 Risoluzione dei problemi

3.6.1 Introduzione

3.6.2 Gestione dei rischi

3.6.2.1 Codifica dei rischi

3.6.2.2 Metriche

3.6.3 Identificazione dei problemi

3.6.4 Strumenti

3.7 Gestione della qualità

3.7.1 Introduzione

3.7.2 Attività

3.7.3 Piano di qualifica

3.7.4 PDCA

3.7.5 Strumenti

4 Processi organizzativi

Lo sviluppo software è un processo complesso e multidisciplinare che richiede una pianificazione, una gestione del tempo e delle risorse accurata, efficiente ed efficace. L'adozione di processi organizzativi ben strutturati è punto cruciale per garantire il successo dello sviluppo software.

4.1 Gestione dei processi

4.1.1 Introduzione

La gestione dei processi si occupa di determinare, migliorare, ottimizzare i processi che fanno da guida alla realizzazione del software. Le attività di gestione dei processi sono:

- **Definizione dei processi**:
  - identificare e documentare i processi chiave coinvolti nello sviluppo software;
  - stabilire le linee guida e procedure per l'esecuzione di ciascun processo;
- **Pianificazione e monitoraggio**:
  - elaborare piani dettagliati per l'esecuzione dei processi;
  - monitorare costantemente l'avanzamento, l'efficacia e la conformità ai requisiti pianificati;
  - stimare i tempi, le risorse ed i costi;
- **Valutazione e miglioramento continuo**
  - condurre valutazioni periodiche dei processi per identificare aree di miglioramento;
  - implementare azioni correttive e preventive per ottimizzare i processi;
- **Formazione e Competenze**
  - assicurare che il personale coinvolto nei processi sia adeguatamente formato;
  - mantenere e sviluppare le competenze necessarie per l'efficace gestione dei processi;
- **Gestione dei rischi**

- identificare e valutare i rischi associati ai processi;
- definire le 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, poiché mira a creare un piano organizzato e coerente per assicurare un'efficace esecuzione delle attività durante l'intero ciclo di vita del software. Il responsabile del progetto assume il compito di coordinare ogni aspetto della pianificazione delle attività, 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 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 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. 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 committenteG e il team, svolgendo il ruolo di mediatore tra le due parti. In particolare si occupa di:

- gestire le relazione con l'esterno;
- pianificare le attività: 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;
- 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 di supporto;

4.1.2.6 Analista

Figura professionale con competenze avanzate riguardo l'attività di analisi dei requisitiG 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 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. Il suo compito primario è implementare il codice conformemente alle specifiche fornite dall'analista e all'architettura definita dal progettista. In particolare, il programmatore:

- scrive codice manutenibile in conformità con le Specifiche Tecniche;
- codifica le varie componenti dell'architettura 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 il lavoro svolto sia conforme alle Norme di progetto;
- verificare che il lavoro svolto sia conforme alle Specifiche Tecniche;
- 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 di tracciamento delle issue (ITS), garantendo così una gestione agevole e trasparente delle attività da svolgere. L'amministratore ha la facoltà di creare e assegnare specifiche issue sulla base delle attività identificate dal responsabile, assicurando chiarezza sulle responsabilità di ciascun membro del team e stabilendo tempi definiti entro cui ciascuna attività deve essere completata. Inoltre, ogni membro del gruppo può monitorare i progressi compiuti nel periodo corrente, consultando lo stato di avanzamento delle varie issue attraverso le Dashboard:

- dashBoard: per una panoramica dettagliata sullo stato delle issue;
- roadMap: per una panoramica temporale dettagliata delle issue;

Procedura per la creazione delle issue:
Le issue vengono create dall'amministratore e devono essere specificati i seguenti attributi:

- Titolo: breve descrizione dell'attività da svolgere.

- Descrizione:
  - descrizione testuale oppure "to-do" tramite bullet points;
  - nell'ultima riga viena specificato il verificatore della issue nel formato: "Verificatore: Nome Cognome";
- assegnatario: membro del team responsabile dell'issue;
- milestone: periodo di riferimento in cui l'attività deve essere completata;
- labels: etichette per categorizzare le issue.
  Per associare ad ogni issue un Configuration Item vengono utilizzati i seguenti label:
  - **NdP**: norme di progetto;
  - **PdP**: piano di progetto
  - **PdQ**: piano di qualifica
  - **AdR**: analisi dei requisiti;
  - **PoC**: proof of concept;
  - **Gls**: glossario;
- milestone: milestone associata alla issue;
- projects: progetti a cui la issue è associata. Se sono presenti dashboard associate ad un progetto, le issue correlate a tale progetto verranno visualizzate nella relativa/e dashboard di progetto;
- development: branch e Pull Request associate alla issue. Quando una Pull Request viene accettata, la relativa issue viene automaticamente chiusa ed eventualmente spostata nella sezione "Done" della dashboard di progetto;

Ciclo di vita di una issue:
Il ciclo di vita è il seguente:

- creazione: l'amministratore crea la issue e la assegna al membro del team responsabile;
- l'amministratore accede alla dashboard di progetto e sposta la issue dalla colonna "No Status" alla colonna "To Do";

- l'assegnatario apre un branch su GitHub seguendo la denominazione suggerita in "Sincronizzazione e Branching";
- quando la issue viene presa in carico dall'assegnatario, questo accede alla DashBoard e sposta la issue dalla colonna "To Do" alla colonna "In Progress";
- una volta che la issue 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";
- all'interno della Dashboard GitHub la issue deve essere spostata dalla colonna "In Progress" alla colonna "Da revisionare";
- il verificatore o i verificatori designati seguono le procedure esposte nella sezione 3.2 per verificare le modifiche apportate al progetto;
- se la verifica ha esito positivo, la issueG viene trasferita dalla colonna "Da revisionare" alla colonna "Done" della Dashboard di GitHub. Nel caso in cui la issue sia associata ad una Pull Request, una volta che quest'ultima viene accettata dal verificatore, la issue viene automaticamente chiusa e spostata nella colonna "Done" della Dashboard di progetto;

4.1.2.11 Strumenti

- GitHub: utilizzato per la condivisione del codice tra i membri del gruppo;
- ClickUp: piattaforma utilizzata per il tracciamento e la gestione delle issue e dei compiti;

4.1.3 Coordinamento

4.1.3.1 Descrizione

Il coordinamento rappresenta l'attività che sovraintende la gestione della comunicazione e la pianificazione degli incontri tra le diverse parti coinvolte in un progetto di ingegneria del software. Questo comprende sia la gestione della comunicazione interna tra i membri del team del progetto, sia la comunicazione esterna con il proponente 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. 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 7Last 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

- **comunicazione sincrone interne**
  Per le comunicazioni sincrone interne, il gruppo 7Last, ha scelto di adottare Discord 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";

- **comunicazione sincrone esterne**
  Per le comunicazioni sincrone esterne,in accordo con l'azienda proponente si è deciso di utilizzare un canale Discord;

4.1.3.4 Comunicazioni asincrone

- **comunicazione asincrone interne**
  Per le comunicazioni asincrone interne, il gruppo 7Last, ha scelto di adottare Telegram in quanto permette di comunicare tramite messaggi di testo, media e file in chat private o come membri di un "gruppo Telegram";

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

  - **email**: per comunicazioni formali e ufficiali;
  - **discord**

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. Generalmente le riunioni sono programmate per ogni venerdi alle ore:

- mercoledì dalle 15 - 16 per riunioni interne;

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, identificano le problematiche riscontrate e discutono di opportunità di miglioramento nei processi 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, 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 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. 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/committente, 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 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, si è stabilito di tenere incontri di stato avanzamento lavori (SAL) con cadenza bisettimanale tramite Google Meet. Durante tali incontri, si affrontano diversi aspetti, tra cui:

- discussione delle attività svolte nel periodo precedente, valutando l'aderenza a quanto concordato e identificando eventuali problematiche riscontrate;
- pianificazione delle attività 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 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 e per le riunioni esterne con il proponente;

- **Telegram**: utilizzato per la comunicazione asincrona interna;
- **Gmail**: come servizio di posta elettronica.

4.1.3.8 Metriche

AGGIUNGERE TABELLA

4.2 Miglioramento

4.2.1 Introduzione

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

- **analisi**: valutazione dei processi per identificare le aree di miglioramento;
- **miglioramento**: attuazione di azioni correttive e preventive per ottimizzare i processi.

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 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 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 standardG 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 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 Gruppo

Sono programmate sessioni formative, condotte dalla proponente, 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 e del software, adotteremo standard internazionali ben definiti per garantire una valutazione rigorosa e conforme agli standard globali. In particolare, l'utilizzo dello standard ISO/IEC 9126 fornirà una solida struttura per valutare la qualità del software, concentrandosi su attributi quali la funzionalità, l'affidabilità, l'usabilità, l'efficienza, la manutenibilità e la portabilità. Questo framework ci consentirà di misurare in modo accurato e completo la qualità del prodotto software . Parallelamente, la suddivisione dei processi in primari, di supporto e organizzativi sarà guidata dall'adozione dello standard ISO/IEC 12207:1995. Infine, l'adozione dello standard ISO/IEC 25010 ci fornirà un quadro completo per la definizione e la suddivisione delle metriche di qualità del software. L'utilizzo congiunto di questi standard consentirà un approccio completo e strutturato alla valutazione della qualità dei processi e del software, assicurando un'elevata coerenza, affidabilità e conformità agli standard riconosciuti a livello internazionale.

5.1 Caratteristiche del sistema, STANDARD CHE USEREMO NOI

5.1.1 Funzionalità

5.1.2 Affidabilità

5.1.3 Usabilità

5.1.4 Efficienza

5.1.5 Manutenibilità

5.1.6 Portabilità

5.2 Suddivisione secondo standard, STANDARD CHE USEREMO NOI

5.2.1 Processi primari

5.2.2 Processi di supporto

5.2.3 Processi organizzativi

6 Metriche di qualità

6.1 Metriche per la qualità di processo

- **Metrica 1M-EV**:
  - **Nome**: Earned Value;
  - **Descrizione**: valore del lavoro effettivamente svolto fino al determinato periodo;
  - **Formula**: $EV = EAC \times \%lavoro\ svolto$;
- **Metrica 2M-PV**:
  - **Nome**: Planned Value;
  - **Descrizione**: stima la somma dei costi realizzativi delle attività imminenti, periodo per periodo;
  - **Formula**: $PV = BAC \times \%lavoro\ svolto$;
- **Metrica 3M-AC**:
  - **Nome**: Actual Cost;
  - **Descrizione**: misura i costi effettivamente sostenuti dall'inizio del progetto fino al presente momento;
  - **Formula**: dato reperibile e costantemente aggiornato in "Piano di Progetto v1.0.0";
- **Metrica 4M-CV**:
  - **Nome**: Cost Variance;
  - **Descrizione**: misura la differenza percentuale di budget tra quanto previsto nella pianificazione di un periodo e l'effettiva realizzazione **CONTROLLARE**;
  - **Formula**: $CV = EV - AC$;
- **Metrica 5M-SV**:
  - **Nome**: Schedule Variance;

  - **Descrizione**: indica in percentuale quanto si è in anticipo o in ritardo rispetto alla pianificazione;
  - **Formula**: $SV = EV - PV$;

- **Metrica 6M-EAC**:
  - **Nome**: Estimated at Completion;
  - **Descrizione**: misura il costo realizzativo stimato per terminare il progetto;
  - **Formula**: $EAC = BAC \div CPI$;

- **Metrica 7M-ETC**:
  - **Nome**: Estimate to Complete;
  - **Descrizione**: Stima dei costi realizzativi fino alla fine del progetto;
  - **Formula**: $ETC = EAC - AC$;

- **Metrica 8M-RSI**:
  - **Nome**: Requirements Stability Index;
  - **Descrizione**: misura impiegata nella quantificazione dell'entità e dell'impatti dei cambiamenti ai requisiti di un progetto;

- **Metrica 9M-SFIN**:
  - **Nome**: Structural Fan In;
  - **Descrizione**: si riferisce ad una classe che è stata progettata in modo tale da essere utilizzata facilmente da molte altre classi;

- **Metrica 10M-SFOUT**:
  - **Nome**: Structural Fan Out;
  - **Descrizione**: numero di moduli subordinati immediati di un metodo;

- **Metrica 13M-CC**:
  - **Nome**: Code Coverage;
  - **Descrizione**: rappresenta il grado in cui il codice sorgente di un programma è testato;

- **Metrica 14M-PTCP**:
  - **Nome**: Passed Test Cases Percentage;
  - **Descrizione**: percentuale di casi di test superati;
  - **Formula**: $PTCP = \frac{Casi\ di\ test\ superati}{Casi\ di\ test\ totali} \times 100$;
- **Metrica 16M-NCR**:
  - **Nome**: Non Calculated Risks;
  - **Descrizione**: indica il numero di rischi non calcolati nel documento di "Analisi dei Requisiti v1.0.0";

6.2 Metriche per la qualità di prodotto

- **Metrica 15M-QMS**:
  - **Nome**: Quality Metrics Satisfied;
  - **Descrizione**: misura che valuta quante metriche, tra quelle definite, sono state implementate e soddisfatte;
  - **Formula**: $QMS = \frac{Metriche\ soddisfatte}{Metriche\ totali} \times 100$;
- **Metrica 17M-TE**:
  - **Nome**: Time Efficiency;
  - **Descrizione**: indicante il livello di efficacia da parte del teamo nello sviluppo di codice di alta qualità;
  - **Formula**: $TE = \frac{Tempo\ impiegato\ per\ lo\ sviluppo}{Tempo\ previsto\ per\ lo\ sviluppo} \times 100$;
- **Metrica 18M-CRO**:
  - **Nome**: Copertura dei Requisiti Obbligatori;
  - **Descrizione**: metrica che valuta quanto del lavoro svolto durante lo sviluppo corrisponda ai requisiti essenziali o obbligatori definiti in fase di analisi dei requisiti;
  - **Formula**: ;
- **Metrica 19M-CRD**:

  - **Nome**: Copertura dei Requisiti Desiderabili;
  - **Descrizione**: valuta quanti di quei requisiti che, se integrati arricchirebbero l'esperienza utente o fornirebbero vantaggi aggiuntivi non strettamente necessari, sono stati implementati o sodisfatti nel prodotto;
  - **Formula**: ;

- **Metrica 20M-CROP**:
  - **Nome**: Copertura dei Requisiti Opzionali;
  - **Descrizione**: valuta quanti dei requisiti aggiuntivi, non essenziali o di bassa priorità, sono stati implementati o soddisfatti nel prodotto;
  - **Formula**: ;

- **Metrica 11M-IG**:
  - **Nome**: Indice Gulpease;
  - **Descrizione**: misura la leggibilità di un testo in base alla lunghezza delle parole e delle frasi;
  - **Formula**: $IG = 89 + \frac{300 \times Numero\,frasi - 10 \times Numero\,lettere}{Numero\,parole}$;

- **Metrica 12M-CO**:
  - **Nome**: Correttezza Ortografica;
  - **Descrizione**: misura la presenza di errori ortografici nei documenti;
  - **Caratteristiche**: affidabilità;

- **Metrica 21M-CC**:
  - **Nome**: Code Coverage;
  - **Descrizione**: fornisce una misura quantitativa del grado o della percentuale di codice eseguito durante i test;

- **Metrica 22M-BC**:
  - **Nome**: Branch Coverage;
  - **Descrizione**: metrica di copertura del codice che indica la percentuale dei rami decisione del codice coperti dai testi;

  - **Formula**: $BC = \frac{Flussi\ funzionali\ testati}{Flussi\ condizionali\ riusciti\ e\ non} \times 100$;

- **Metrica 23M-SC**:
  - **Nome**: Statement Coverage;
  - **Descrizione**: metrica di copertura del codice che indica la percentuale degli statement del codice coperti dai test;
  - **Formula**: $SC = \frac{Statement\ testati}{Statement\ totali} \times 100$;

- **Metrica 24M-FD**:
  - **Nome**: Failure Density;
  - **Descrizione**: misura che indica il numero di difetti trovati in un software o in una parte di esso durante il ciclo di sviluppo;

- **Metrica 25M-FU**:
  - **Nome**: Facilità di Utilizzo;
  - **Descrizione**: metrica che misura l'usabilità di un sistema software;
  - **Caratteristiche**: usabilità;

- **Metrica 26M-TA**:
  - **Nome**: Tempo di Apprendimento;
  - **Descrizione**: misura il tempo massimo richiesto per apprendere l'utilizzo del prodotto;
  - **Caratteristiche**: uasbilità;

- **Metrica 27M-UR**:
  - **Nome**: Utilizzo Risorse;
  - **Descrizione**: ;
  - **Formula**: ;

- **Metrica 28M-CCM**:
  - **Nome**: Complessità Ciclomatica;

  - **Descrizione**: rappresenta la complessità di un metodo in base ai percorsi possibili **CONTROLLARE**;
  - **Formula**: $CCM = e - n + 2$;
  - **Legenda**:
    * e: numero di archi del grafo del flusso di esecuizione del metodo;
    * n: numero di nodi del grafo del flusso di esecuzione del metodo.
  - **Caratteristiche**: manutenibilità;

- **Metrica 29M-CSM**:
  - **Nome**: Code Smell;
  - **Descrizione**: livello di pulizia e manutenibilità del codice sviluppato;

- **Metrica 30M-COC**:
  - **Nome**: Coefficient of Coupling;
  - **Descrizione**: rappresenta il grado di dipendenza tra diversi moduli o componenti di un sistema software;
  - **Formula**: $COC = \frac{Numero\ di\ dipendenze}{Numero\ di\ moduli}$;