Norme di Progetto

v1.0

7Last

Versioni

Ver.	Data	Autore	Descrizione
1.0	2024-05-12	Leonardo Baldo	Verifica finale
0.3	2024-04-05	Raul Seganfreddo	Aggiunta Introduzione
0.2	2024-04-05	Matteo Tiozzo	Modificato tabella versioni
0.1	2024-03-29	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 ogni membro del gruppo è tenuto a rispettare durante lo svolgimento del progetto. Lo scopo quindi è quello di definire il *Way of Working* del team, in modo da garantire un lavoro efficiente e di qualità. La stesura di questo documento avverrà nelle prime fasi dello svolgimento del progetto, in modo tale da aggiornarsi e adattarsi alle esigenze del gruppo.
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 città smart, in grado di raccogliere e analizzare in tempo reale dati provenienti da diverse fonti, come sensori, dispositivi indossabili e macchine. La piattaforma avrà i seguenti scopi:

- **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** i big data: sarà in grado di gestire grandi volumi di dati provenienti da diverse tipologie di sensori, 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. Il progetto mira dunque a creare una piattaforma che sia **efficiente**, **efficace** e **accessibile**.

1.3 Glossario

Per evitare ambiguità e facilitare la comprensione del documento, si farà uso di un *glossario* contenente la definizione dei termini tecnici e degli acronimi utilizzati. I termini di questo documento presenti nel glossario saranno riconoscibili in quanto accompagnati da una "G" a pedice e aventi il link alla relativa definizione nell'apposito documento online.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato d'appalto C6 SyncCity: a 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
  `https://7last.github.io/docs/rtb/documentazione-interna/glossario`
- Documentazione Git
  `https://git-scm.com/docs`
- Documentazione Latex
  `https://www.latex-project.org/help/documentation/`
- Documentazione Python
  `https://docs.python.org/3/`
- Documentazione Kafka
  `https://kafka.apache.org/documentation/`
- Documentazione ClickHouse
  `https://clickhouse.com/docs`
- Documentazione Grafana
  `https://grafana.com/docs/grafana/latest/`

- Documentazione Docker
  `https://docs.docker.com/`

2 Processi primari

2.1 Fornitura

2.1.1 Introduzione

Il processo di fornitura è un percorso strutturato che stabilisce un accordo contrattuale tra il fornitore e il cliente e guida lo sviluppo e la consegna di un prodotto software, dalla concezione iniziale fino alla manutenzione post-rilascio. Questo processo è fondamentale per garantire che il software soddisfi i requisiti del cliente, sia di alta qualità e venga consegnato nei tempi e nei costi previsti.

2.1.2 Attività

Il processo di fornitura comprende le seguenti fasi:

- **preparazione della proposta**: questa fase iniziale prevede la raccolta di informazioni e la stesura di una proposta formale per il cliente. Si suddivide in:
  - analisi delle esigenze del cliente;
  - studio di fattibilità;
  - elaborazione della proposta;
- **contrattazione**: durante questa fase il fornitore e il cliente discutono e negoziano i termini del contratto. Alcune attività tipiche di questa fase includono:
  - discussione dei termini e delle condizioni;
  - stesura e revisione del contratto;
  - firma del contratto;
- **pianificazione**: fondamentale per organizzare e programmare le attività del progetto. Questa fase include:
  - stesura delle milestones;
  - stesura del piano di progetto;
  - assegnazione dei compiti e delle risorse;
- **esecuzione**: si effettua la realizzazione concreta del progetto, con la costruzione del prodotto software. L'attività è costituita principalmente da:

  - sviluppo del software;
  - test e verifica;
  - documentazione;

- **revisione**: prevede una valutazione approfondita del lavoro svolto per garantire che tutto sia conforme agli standard di qualità e ai requisiti contrattuali. Questa fase include:
  - revisione del codice;
  - test di accettazione;
  - correzione delle discrepanze;

- **consegna**: consiste nella consegna del prodotto finale al cliente e nella preparazione per il supporto post-rilascio Si compone di:
  - consegna del software;
  - formazione del personale;
  - supporto post-rilascio;

2.1.3 Comunicazioni con l'azienda proponente

SyncLab S.r.l. offre un indirizzo email e un canale Discord per la comunicazione tramite messaggi, oltre a Google Meet per gli incontri telematici. Gli incontri online si terranno inizialmente con cadenza bisettimanale, con la possibilità di organizzare incontri extra su richiesta del gruppo. Per ogni colloquio con l'azienda proponente sarà redatto un verbale che riassumerà i temi trattati. Tali documenti saranno disponibili nella repository del team *7Last*.

2.1.4 Documentazione fornita

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

2.1.4.1 Piano di qualifica

Il *Piano di Qualifica* è un documento che specifica le responsabilità e le attività del verificatore nel progetto, delineando le strategie e gli approcci adottati per garantire la qualità del prodotto software in fase di sviluppo. Redatto dall'amministratore, questo documento descrive le modalità di verifica e validazione, nonché gli standard e le procedure di qualità da seguire durante l'intero ciclo di vita del progetto. Tutti i membri del team di progetto si riferiranno a questo documento per assicurarsi di raggiungere la qualità desiderata. Questo documento è organizzato in diverse sezioni, tra cui:

- **qualità di processo**: definisce gli standard e le procedure adottate durante il progetto al fine di garantire che i processi seguiti siano di alta qualità;
- **qualità di prodotto**: stabilisce i criteri, le metriche e gli standard da rispettare affinchè il prodotto finale sia di alta qualità;
- **specifica dei test**: contiene la descrizione di tutti i test necessari a verificare che il prodotto soddisfi i requisiti specificati;
- **cruscotto 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.

2.1.4.2 Analisi dei requisiti

L'*Analisi dei Requisiti* non solo aiuta a definire e comprendere le esigenze e le aspettative degli stakeholder in relazione al prodotto software in fase di sviluppo, ma rappresenta anche un punto di riferimento fondamentale che permea l'intero ciclo di vita del progetto. Questo documento fornisce una panoramica chiara delle richieste degli utenti finali e degli altri soggetti coinvolti nel progetto. Serve come guida per il team di sviluppo, fornendo una mappa dettagliata delle funzionalità da implementare e dei requisiti da soddisfare. Infine, durante le fasi di verifica e validazione, questo documento

viene utilizzato come base di riferimento per garantire che il prodotto software sviluppato rispetti le richieste e le necessità degli utenti finali, contribuendo così al successo complessivo del progetto. Il documento è composto principalmente da:

- **descrizione**: fornisce una panoramica generale del progetto, del contesto in cui il software verrà utilizzato e degli obiettivi principali. Include una spiegazione chiara dello scopo del documento e delle sue finalità, fornendo un quadro complessivo del progetto e del suo contesto;
- **elenco dei casi d'uso**: qui vengono riportati tutti gli scenari possibili in cui il sistema software potrebbe essere utilizzato dagli utenti finali. Ogni caso d'uso descrive dettagliatamente le azioni che gli utenti compiono nel sistema, permettendo di identificare requisiti non ovvi inizialmente. Questa sezione fornisce una visione dettagliata delle interazioni previste tra gli utenti e il sistema, contribuendo a definire le funzionalità e le caratteristiche del software;
- **elenco dei requisiti**: in questa sezione vengono specificati tutti i vincoli richiesti dal proponente o dedotti in base all'analisi dei casi d'uso associati ad essi. I requisiti delineano le funzionalità, le prestazioni, le restrizioni e altri aspetti critici del software che devono essere soddisfatti. Essi rappresentano le basi su cui il team progetta e implementa il software, assicurando che il prodotto finale rispetti le aspettative degli stakeholder e soddisfi le necessità degli utenti finali.

2.1.4.3 Piano di progetto

Il *Piano di Progetto* delinea in modo dettagliato e organizzato tutti gli aspetti e le attività coinvolte nella gestione di un progetto di sviluppo del software. Esso funge da strumento di pianificazione, monitoraggio e controllo, offrendo una roadmap chiara e ben definita per il team di progetto
Comprende le seguenti informazioni:

- **analisi dei rischi**: identificazione delle potenziali sfide che potrebbero insorgere durante il processo e che potrebbero arrecare ritardi e/o impedimenti al progredire del progetto. Il gruppo si impegna a fornire soluzioni per tali problemi il prima possibile. I rischi sono classificati in tre categorie principali: rischi organizzativi, rischi tecnologici e rischi comunicativi;
- **modello di sviluppo**: illustra l'approccio organizzato e metodologico implementato dal team per sviluppare il prodotto;

- **pianificazione temporale**: definisce i periodi temporali con eventi e attività correlate, all'interno di un calendario. Per ciascun periodo, saranno specificati i compiti dei vari ruoli e una stima del tempo richiesto da ciascun membro del team per completare le rispettive attività;
- **preventivo**: valuta la durata prevista di ciascun periodo, indicando il tempo necessario per poter portare a termine tutte le attività pianificate;
- **revisione dell'avanzamento**: analizza il lavoro effettivamente svolto rispetto alle previsioni, al fine di ottenere uno stato di avanzamento del progetto al termine di ogni periodo.

2.1.4.4 Glossario

Il *Glossario* è un elenco dettagliato e organizzato di termini, acronimi e definizioni utilizzati nel contesto del progetto. Fornisce una chiara comprensione dei concetti e dei termini specifici impiegati nel progetto, garantendo una comunicazione efficace e coesa tra tutti i membri del team, nonché con gli stakeholder esterni.

2.1.4.5 Lettera di presentazione

La *Lettera di Presentazione* costituisce il mezzo attraverso il quale il gruppo *7Last* espone il proprio interesse a partecipare attivamente alla fase di revisione del prodotto software. Questo documento non solo mette in luce la disponibilità della documentazione rilevante per i committenti e il proponente, ma stabilisce anche 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:

- **Discord**: come piattaforma per le riunioni interne e come un metodo informale per contattare l'azienda proponente tramite messaggistica;
- **Google Meet**: utilizzato per le riunioni con la proponente;
- **LaTeX**: un sistema di preparazione di documenti utilizzato principalmente per la creazione di documenti tecnici e scientifici;

- **GitHub**: un servizio di hosting per progetti software che offre funzionalità di controllo di versione e collaborazione;

2.2 Sviluppo

2.2.1 Introduzione

L'ISO/IEC 12207:1995 fornisce un quadro completo e strutturato per la gestione del ciclo di vita del software. Questo standard definisce i processi, le attività e le mansioni coinvolte nello sviluppo, nell'acquisizione e nella manutenzione del software. È fondamentale completare tali operazioni in stretto rispetto alle direttive e ai requisiti definiti nel contratto con il cliente, assicurando quindi una realizzazione precisa e in linea con le specifiche richieste.

2.2.2 Analisi dei requisiti

2.2.2.1 Descrizione

L' *analisi dei requisiti* ha lo scopo di identificare, definire e documentare in modo esaustivo le necessità, le funzionalità e le prestazioni che il sistema software deve soddisfare. Questo documento rappresenta il punto di partenza cruciale per il processo di sviluppo del software, fornendo una base solida e chiara per la progettazione, l'implementazione e la verifica del sistema. Attraverso l'analisi dei requisiti, si cerca di comprendere appieno le esigenze degli stakeholder, inclusi utenti finali, clienti e altri soggetti coinvolti nel progetto, al fine di garantire che il prodotto software sviluppato soddisfi le loro aspettative e necessità. L'analisi dei requisiti include tipicamente la raccolta e la documentazione dei requisiti funzionali e non funzionali, la definizione dei casi d'uso, la modellazione dei dati e dei processi di business, nonché la prioritizzazione e la tracciabilità dei requisiti lungo l'intero ciclo di vita del software. Inoltre, questo documento fornisce un punto di riferimento essenziale per tutti i membri del team di sviluppo, consentendo loro di concentrarsi sulle esigenze del cliente e di mantenere l'allineamento con gli obiettivi e le aspettative del progetto. Questa attività richiede di rispondere a domande fondamentali come "Quali sono i vincoli tecnici, legali o di altro tipo?", "Quali sono i casi d'uso e gli scenari di utilizzo del sistema?".

2.2.2.2 Scopo

- Definire gli obiettivi del prodotto al fine di soddisfare le aspettative;
- 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

Agli analisti il compito di redigere l'*analisi dei requisiti*, comprendendo i seguenti elementi:

- **introduzione**: presentazione e scopo del documento stesso;
- **descrizione**: analisi approfondita del prodotto, includendo:
  - obiettivi 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;
    - diagrammi 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-[identificativo_caso_principale].[identificativo_sotto_caso]

   - **Identificativo sotto caso**: 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. **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).

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

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

6. **User story**: 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

offrono una rappresentazione visiva delle interazioni tra gli utenti finali (o attori) e il sistema software, delineando i vari scenari in cui il sistema verrà utilizzato. Ogni caso d'uso descrive una sequenza specifica di azioni che gli attori compiono per raggiungere un obiettivo particolare con il sistema. I diagrammi dei casi d'uso aiutano a identificare i requisiti funzionali del sistema, fornendo una comprensione chiara e condivisa delle aspettative degli utenti. Inoltre, facilitano la comunicazione tra gli sviluppatori e gli stakeholder, assicurando che tutte le funzionalità necessarie siano considerate e correttamente implementate nel software. Di seguito vengono elencati i principali componenti di un diagramma dei casi d'uso:

- **Attori**: rappresentano gli utenti o i sistemi esterni che interagiscono con il sistema software. Gli attori possono essere persone, altri sistemi software o dispositivi che utilizzano le funzionalità del sistema. Sono rappresentati come figure stilizzate.

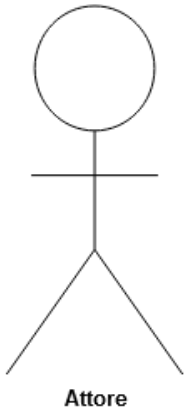

Figura 1: Diagramma dei casi d'uso - Attore

- **Casi d'uso**: rappresentano le funzionalità o i servizi offerti dal sistema che producono un risultato di valore per un attore. Ogni caso d'uso descrive una sequenza specifica di interazioni tra gli attori e il sistema. Sono rappresentati come ovali.

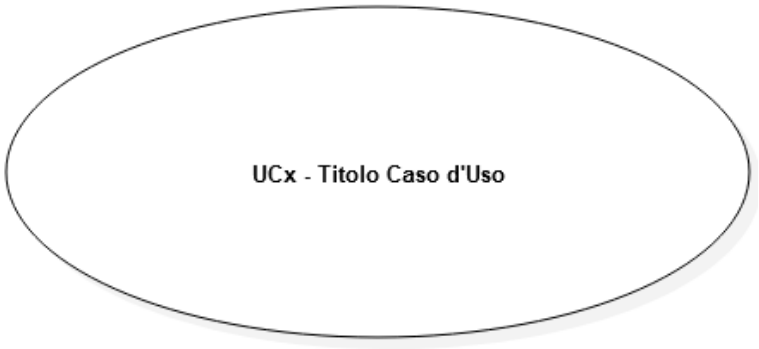

Figura 2: Diagramma dei casi d'uso - Caso d'uso

- **Sottocasi d'uso**: rappresentano scenari specifici e dettagliati che si verificano all'interno di un caso d'uso principale. Questi aiutano a modularizzare e riutilizzare le funzionalità comuni e a gestire le variazioni nei processi.

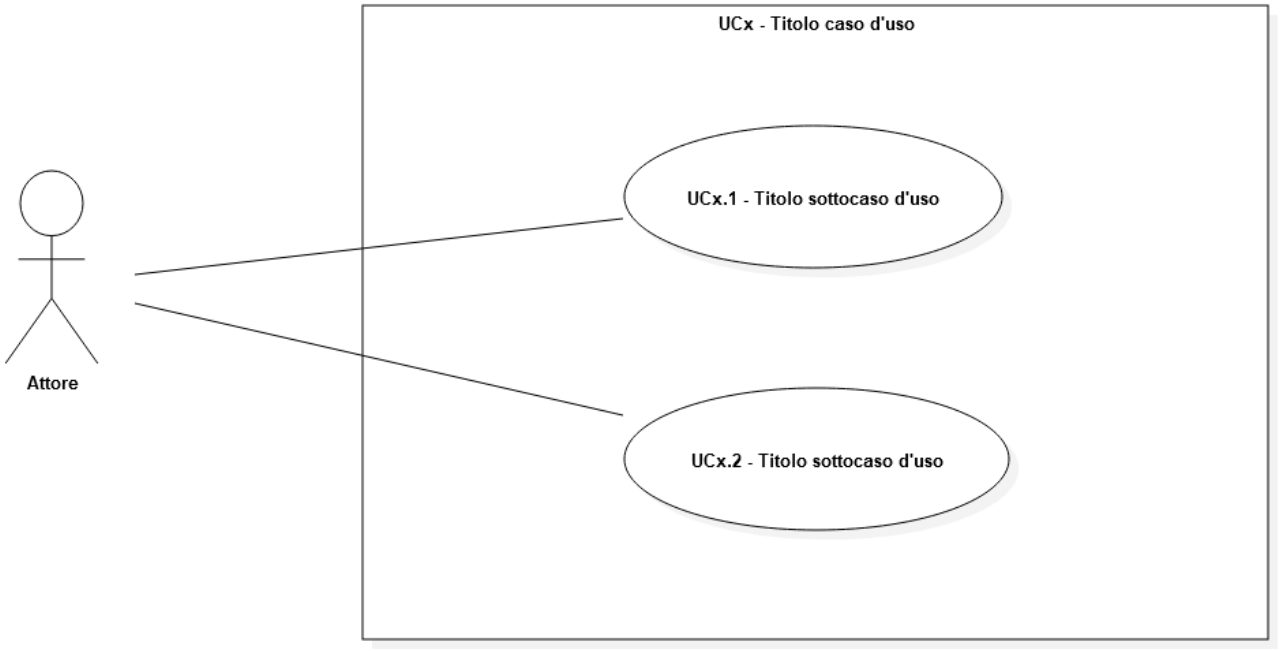

Figura 3: Diagramma dei casi d'uso - Sottocaso d'uso

- **Relazioni tra Attori e Casi d'Uso**
  - **Associazione**: è la relazione fondamentale che collega un attore a un caso d'uso, indicando che l'attore interagisce con il sistema per eseguire quella specifica funzione. Viene rappresentata da una linea semplice che collega l'attore al caso d'uso.

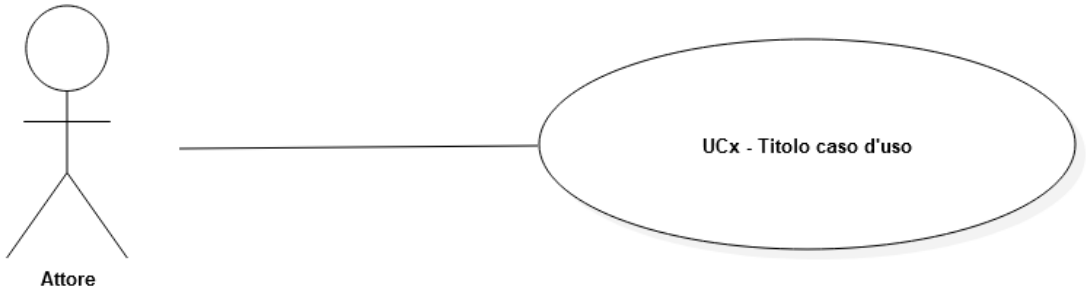

Figura 4: Diagramma dei casi d'uso - Associazione

- **Relazioni tra Attori**
  - **Generalizzazione**: rappresenta una relazione ereditaria in cui un attore figlio eredita il comportamento di un attore genitore. Questo viene utilizzato quando diversi attori condividono comportamenti comuni. Viene rappresentata da una freccia con una linea piena che punta dall'attore figlio all'attore genitore.

Figura 5: Diagramma dei casi d'uso - Generalizzazione tra attori

- **Relazioni tra Casi d'Uso**
  - **Inclusione**: rappresenta una relazione in cui un caso d'uso (incluso) è utilizzato come parte di un altro caso d'uso (principale). Il caso d'uso incluso contiene funzionalità che sono riutilizzate in più casi d'uso principali, permettendo di evitare la duplicazione di comportamenti comuni. È rappresentata da una freccia tratteggiata con l'etichetta «include» che parte dal caso d'uso principale e punta verso il caso d'uso incluso.

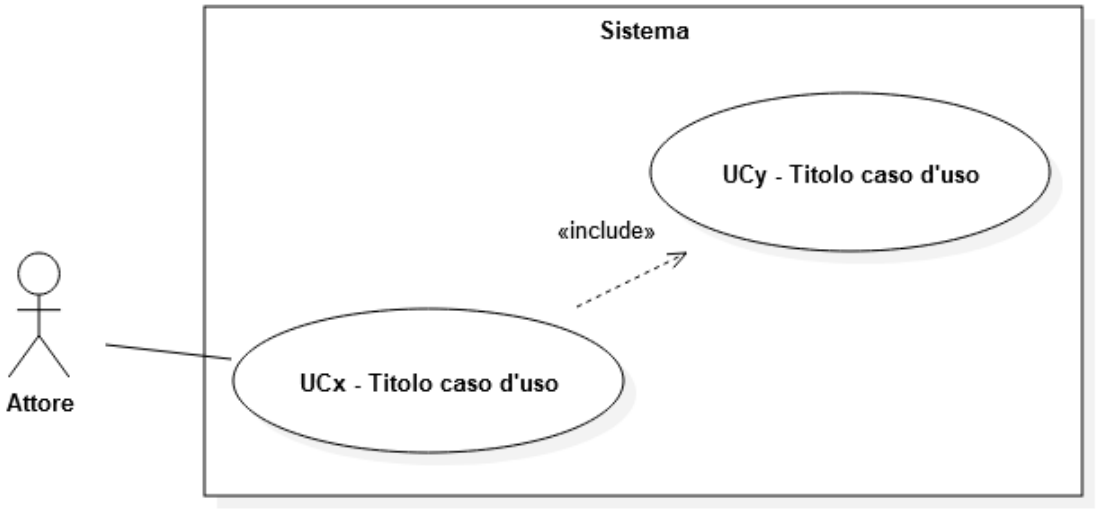

Figura 6: Diagramma dei casi d'uso - Inclusione

– **Estensione**: mostra una relazione in cui un caso d'uso (esteso) aggiunge comportamento opzionale o condizionale a un altro caso d'uso (principale). Il caso d'uso esteso viene eseguito solo sotto determinate condizioni o su richiesta. L'estensione è rappresentata da una freccia tratteggiata con l'etichetta «extend» che parte dal caso d'uso esteso e punta verso il caso d'uso principale.

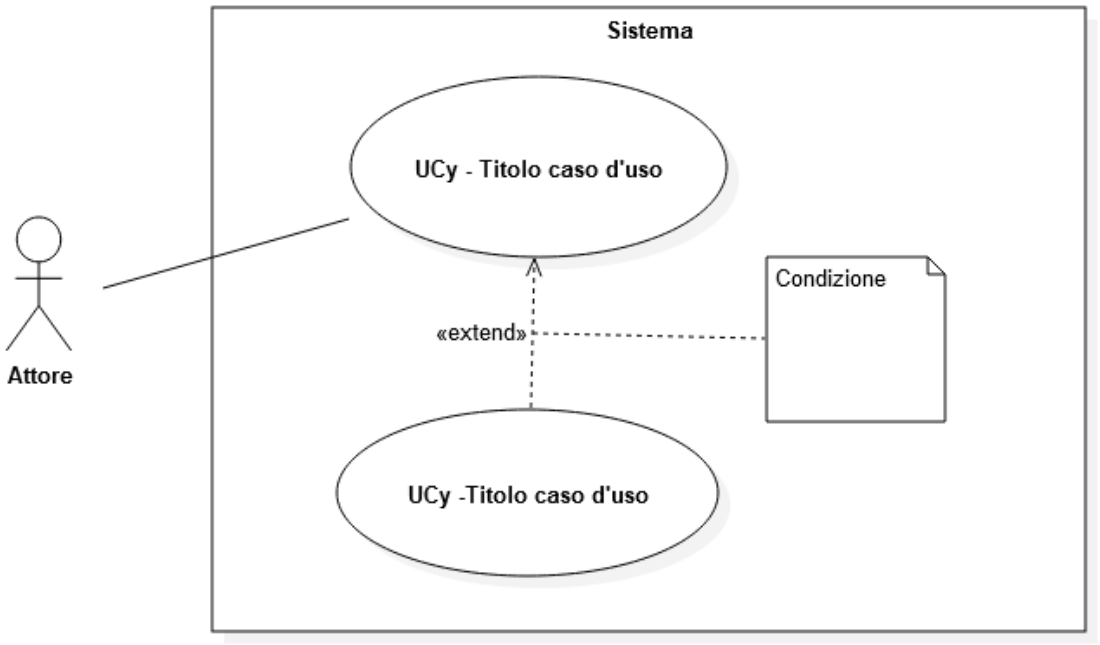

Figura 7: Diagramma dei casi d'uso - Estensione

- **Generalizzazione casi d'uso**: rappresenta una relazione ereditaria in cui un caso d'uso figlio eredita il comportamento di un caso d'uso genitore. Questo è utile per descrivere variazioni o estensioni di un comportamento base. È rappresentata da una freccia con una linea piena che punta dal caso d'uso figlio al caso d'uso genitore.

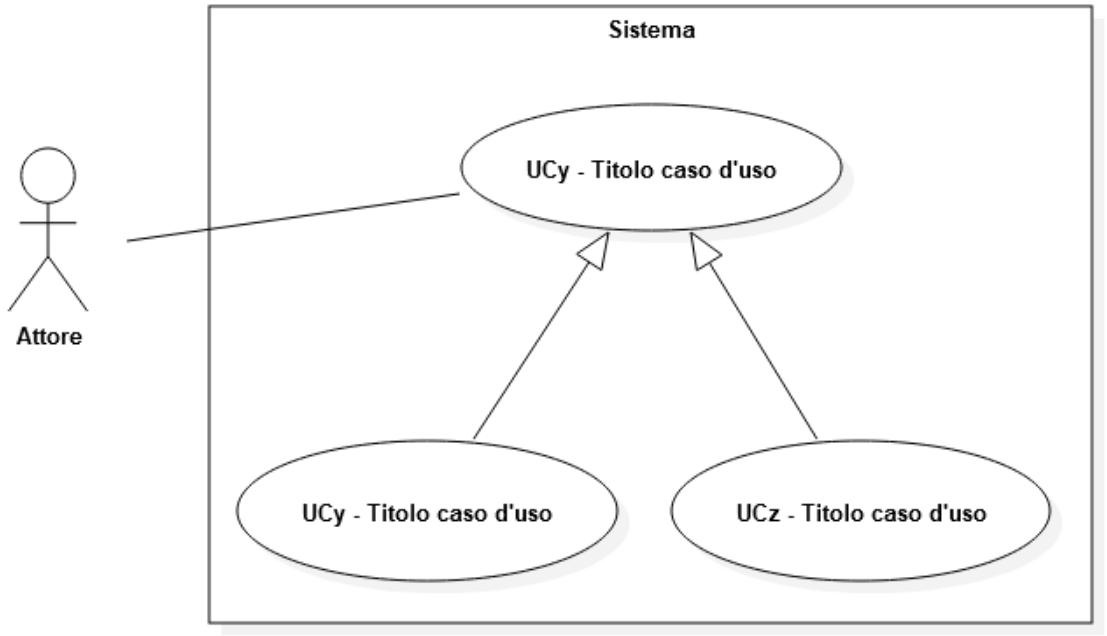

Figura 8: Diagramma dei casi d'uso - Generalizzazione tra casi d'uso

- **Sistema**: delimita i confini del sistema software, indicando quali funzionalità sono incluse e quali sono esterne al sistema. Il sistema è rappresentato come un rettangolo che racchiude i casi d'uso.

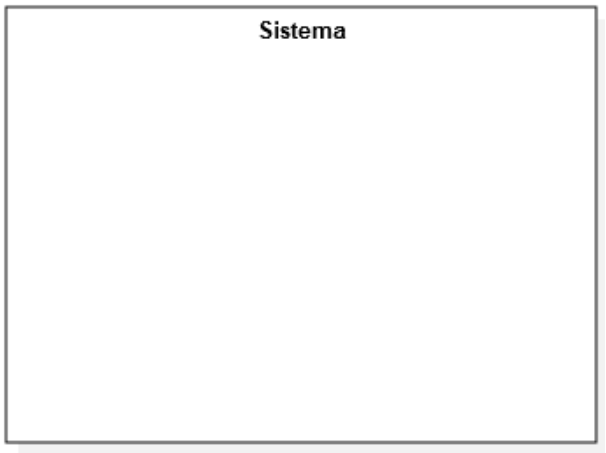

Figura 9: Diagramma dei casi d'uso - Sistema

2.2.2.6 Requisiti

Delineano una descrizione dettagliata delle funzionalità e delle caratteristiche che il software deve possedere per soddisfare le esigenze degli utenti e degli stakeholder. Questi requisiti costituiscono la base per tutto il ciclo di sviluppo del software e sono fondamentali per il successo del progetto. Si suddividono generalmente in due categorie principali:

- **requisiti funzionali**, i quali descrivono le azioni specifiche che il sistema deve essere in grado di eseguire. Rispondono alla domanda "cosa deve fare il sistema?";
- **requisiti non funzionali**, che definiscono le proprietà e le caratteristiche del sistema che ne influenzano il funzionamento e l'uso. Rispondono alla domanda "come deve comportarsi il 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. **codice**: i requisiti sono codificati nel seguente modo:

   R[Tipologia]-[Codice]

   dove

   - **[Tipologia]**: indica la tipologia del requisito, che può essere:
     - **F**: requisito funzionale;
     - **Q**: requisito di qualità;
     - **V**: requisito di vincolo.
   - **[Codice]**: è un numero progressivo che identifica univocamente il requisito.
2. **Importanza**: indica il grado di importanza del requisito, che può essere:
   - **Obbligatorio**: irrinunciabile per il committente$_G$;
   - **Desiderabile**: non strettamente necessario, ma che porta valore aggiunto al prodotto;
   - **Opzionale**: relativo a funzionalità aggiuntive.
3. **Fonte**: indica la fonte da cui è stato identificato il requisito, che può essere:

- **capitolato**$_G$: requisiti individuati a seguito dell'analisi dello stesso;
- **interno**: requisiti individuati durante le riunioni interne e da coloro che hanno il ruolo di analista;
- **esterno**: requisiti aggiuntivi individuati in seguito a incontri con la proponente$_G$;
- **piano di qualifica**$_G$: requisiti necessari per adeguare il prodotto agli standard di qualità definiti nel documento *Piano di Qualifica*$_G$;
- **norme di progetto**$_G$: requisiti necessari per adeguare il prodotto alle norme stabilite nel documento *Norme di Progetto*$_G$.

2.2.2.7 Metriche

Nell'analisi dei requisiti, le metriche sono strumenti utilizzati per misurare vari aspetti del processo di sviluppo e del prodotto finale. All'interno di un'analisi dei requisiti, le metriche svolgono un ruolo cruciale nel garantire che il software sia sviluppato in modo efficiente, soddisfi gli standard di qualità attesi e adempia alle specifiche richieste dagli stakeholder.

Metrica	Nome completo
0M-CRO	Copertura dei Requisiti Obbligatori.
1M-CRD	Copertura dei Requisiti Desiderabili.
2M-CROP	Copertura dei Requisiti Opzionali.

Tabella 1: Metriche per l'analisi dei requisiti

2.2.2.8 Strumenti

StarUML è uno strumento versatile e potente per la modellazione software, essenziale per chiunque desideri adottare un approccio strutturato e visivo alla progettazione e sviluppo di sistemi software complessi. Con il suo supporto per UML e altre notazioni di modellazione, le capacità di estensibilità e collaborazione, e la generazione automatica di documentazione, StarUML rappresenta una risorsa preziosa per migliorare la qualità e l'efficienza dei progetti software.

2.2.3 Progettazione

2.2.3.1 Descrizione

L'attività di progettazione svolge un ruolo fondamentale nello sviluppo del software, consentendo di definire l'architettura, i componenti e le interfacce del sistema in modo da soddisfare i requisiti definiti durante l'analisi. Questa fase si basa sull'analisi dei requisiti e ha lo scopo di tradurre i requisiti funzionali e non funzionali in una soluzione tecnica implementabile.

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:

- **definizione dell'architettura**: in questa fase si definisce l'architettura del sistema, determinando la struttura generale, i componenti principali e le relazioni tra di essi. Questo include la scelta delle tecnologie, dei linguaggi di programmazione e delle piattaforme da utilizzare;
- **suddivisione in moduli**: si suddivide il sistema in moduli o componenti più piccoli e gestibili, ciascuno con responsabilità specifiche. Questa suddivisione facilita lo sviluppo, il testing e la manutenzione del software;
- **definizione delle interfacce**: si specificano le interfacce tra i diversi moduli o componenti del sistema, stabilendo come essi comunicano e interagiscono tra di loro. Questo garantisce una corretta integrazione e interoperabilità del sistema;
- **identificazione delle risorse**: vengono identificate le risorse necessarie per implementare il sistema, come hardware, software, risorse umane e finanziarie. Questo aiuta a pianificare e gestire le risorse in modo efficace durante lo sviluppo;
- **valutazione delle prestazioni**: si valutano le prestazioni del sistema, identificando potenziali problemi di scalabilità, affidabilità e prestazioni. Questo consente di ottimizzare il design per garantire che il sistema soddisfi i requisiti di prestazione.

2.2.3.3 Documentazione

Specifica tecnica
Fornisce una guida dettagliata ed esaustiva per la progettazione e lo sviluppo del sistema. In essa sono delineate con precisione l'architettura tecnica del software, con tutti i suoi componenti e interfacce, le funzionalità che il sistema dovrà offrire e le tecnologie che saranno impiegate per realizzarlo. Questo documento costituisce il punto di partenza per l'intero ciclo di sviluppo del software, fornendo una base solida e strutturata su cui il team di sviluppo potrà lavorare in modo efficace ed efficiente. Tra gli elementi chiave inclusi in questo documento vi sono:

- **architettura del sistema**: questa sezione delinea l'architettura generale del sistema software, identificando i principali componenti, i moduli e le interfacce. Include anche informazioni sull'organizzazione logica e fisica del sistema, come ad esempio la suddivisione in livelli o strati e le relazioni tra di essi;
- **tecnologie utilizzate**: specifica le tecnologie, i linguaggi di programmazione, i framework e le librerie che saranno impiegate nello sviluppo del sistema. Questo include anche eventuali tool e ambienti di sviluppo adottati;
- **struttura dei dati**: descrive la struttura e la gestione dei dati all'interno del sistema, inclusi database, file system, protocolli di accesso ai dati e modelli di persistenza;
- **interfacce esterne**: indica le interfacce con altri sistemi o applicazioni esterne con cui il sistema deve interagire. Questo include anche la definizione dei formati dei dati scambiati e i protocolli di comunicazione utilizzati;
- **design pattern**: descrizione dei design pattern utilizzati per risolvere problemi comuni e ricorrenti durante lo sviluppo del software;
- **pianificazione e risorse**: fornisce una pianificazione dettagliata del lavoro da svolgere per implementare la specifica tecnica, inclusi tempi, costi e risorse necessarie;
- **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;

- **sicurezza e privacy**: specifica i requisiti e le misure di sicurezza che saranno implementati nel sistema per proteggere i dati sensibili e garantire la privacy degli utenti.

2.2.3.4 Qualità dell'architettura

- **Scalabilità**: rappresenta la capacità del sistema di gestire un aumento del carico di lavoro o delle risorse senza compromettere le prestazioni o la qualità del servizio. Un'architettura scalabile può adattarsi dinamicamente alle variazioni delle richieste degli utenti e delle risorse disponibili;
- **flessibilità**: capacità del sistema di adattarsi ai cambiamenti nei requisiti o nell'ambiente operativo senza richiedere modifiche significative. Un'architettura flessibile è caratterizzata da componenti ben separati e interfacce standardizzate, che consentono una facile modifica e l'aggiunta di nuove funzionalità;
- **sicurezza**: protezione dei dati, delle risorse e del sistema da minacce esterne e interne. Un'architettura sicura include meccanismi di autenticazione, autorizzazione, crittografia e controllo degli accessi per garantire la riservatezza, l'integrità e la disponibilità delle informazioni;
- **manutenibilità**: ovvero la facilità di comprendere, modificare e correggere il software nel corso del suo ciclo di vita. Un'architettura manutenibile è caratterizzata da un codice ben strutturato, documentato e modularizzato, che consente agli sviluppatori di individuare e risolvere rapidamente eventuali problemi;
- **testabilità**: facilità di testare il software per garantire che soddisfi i requisiti funzionali e non funzionali. Un'architettura testabile include componenti ben isolati e interfacce chiare che consentono l'automazione dei test e la verifica continua della qualità del software;
- **affidabilità**: garanzia del corretto funzionamento del sistema in condizioni normali e anomale. Un'architettura affidabile include meccanismi di gestione degli errori, di recupero e di ripristino che consentono al sistema di continuare a operare in modo affidabile anche in presenza di guasti o interruzioni;
- **performance**: capacità del sistema di fornire risposte rapide e tempi di elaborazione efficienti anche in presenza di carichi di lavoro elevati. Un'architettura perfor-

mante include ottimizzazioni del codice, della gestione delle risorse e dell'accesso ai dati per massimizzare le prestazioni del sistema;

- **usabilità**: facilità di utilizzo e comprensione del sistema da parte degli utenti finali. Un'architettura usabile include un'interfaccia utente intuitiva, una struttura di navigazione chiara e una presentazione coerente delle informazioni per garantire un'esperienza utente positiva;
- **compatibilità**: capacità del sistema di interoperare con altri sistemi, piattaforme o tecnologie senza problemi. Un'architettura compatibile include standard aperti, interfacce ben definite e protocolli di comunicazione standardizzati che consentono l'integrazione con sistemi esterni;
- **documentazione**: fornitura di documentazione completa e accurata sull'architettura del sistema, inclusi diagrammi, specifiche tecniche, manuale utente e guide per gli sviluppatori. Una buona documentazione facilita la comprensione, la manutenzione e l'evoluzione del sistema nel tempo;
- **safety**: l'architettura deve garantire la sicurezza del sistema, in modo che possa proteggere i dati e le informazioni sensibili in seguito a malfunzionamenti.

2.2.3.5 Diagrammi UML

Vantaggi:

- **chiarezza visiva**: forniscono una rappresentazione visuale chiara e intuitiva delle relazioni e delle interazioni tra gli elementi del sistema software, facilitando la comprensione del sistema da parte degli stakeholder;
- **standardizzazione**: UML è uno standard riconosciuto a livello internazionale per la modellazione dei sistemi software, il che significa che i diagrammi UML sono facilmente comprensibili da parte di professionisti del settore in tutto il mondo;
- **comunicazione efficace**: forniscono un linguaggio comune per la comunicazione tra gli sviluppatori, gli stakeholder tecnici e non tecnici e altri membri del team, consentendo una comunicazione più efficace e una condivisione delle informazioni più chiara;

- **analisi e progettazione**: possono essere utilizzati durante le fasi di analisi e progettazione del ciclo di sviluppo del software per visualizzare e analizzare i requisiti, le relazioni tra i componenti e le interazioni del sistema;
- **modellazione e requisiti**: UML consente di modellare i requisiti del sistema in modo chiaro e strutturato, identificando casi d'uso, scenari e vincoli che guidano lo sviluppo del software;
- **facilità di manutenzione**: è più facile comprendere l'architettura del sistema e individuare eventuali problemi o aree di miglioramento, facilitando così la manutenzione e l'evoluzione del software nel tempo;
- **supporto agli standard di progettazione**: I diagrammi UML possono essere utilizzati per applicare e comunicare i principi di progettazione software ben consolidati, come l'incapsulamento, l'ereditarietà e il polimorfismo;
- **documentazione**: possono essere utilizzati per generare documentazione tecnica dettagliata sul sistema software, fornendo una guida completa per gli sviluppatori, gli utenti finali e gli altri stakeholder.

A supporto della progettazione, il team utilizzerà i seguenti **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

Essenziali per rappresentare le interazioni e le dipendenze tra le varie componenti del sistema software, queste relazioni forniscono un quadro completo della struttura e del comportamento del sistema, consentendo agli sviluppatori di comprendere meglio come le classi si relazionano tra loro e di progettare un sistema coerente e ben strutturato. Di seguito sono descritte le principali relazioni tra le classi:

- **Associazione**: rappresenta una relazione tra due classi, indicando che un'istanza di una classe è correlata a un'istanza di un'altra classe. È raffigurata da una linea solida tra le classi coinvolte. La linea può includere etichette per indicare il nome dell'associazione, nonché molteplicità e ruoli per specificare la cardinalità e il ruolo delle classi nell'associazione.

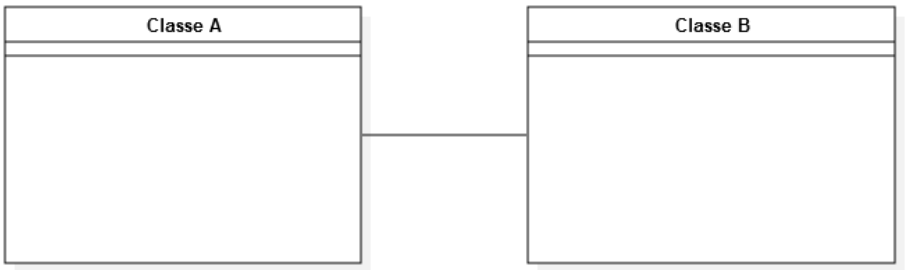

Figura 10: Diagramma delle classi - Associazione

- **Aggregazione**: indica una relazione "tutto-parte" tra due classi, dove una classe è composta da una o più istanze di un'altra classe, ma le istanze possono esistere indipendentemente dalla classe principale. Composta da una linea con una freccia vuota che punta dalla parte composta (parte) al tutto (oggetto principale). La linea può includere un rombo vuoto sulla parte composta per indicare l'aggregazione.

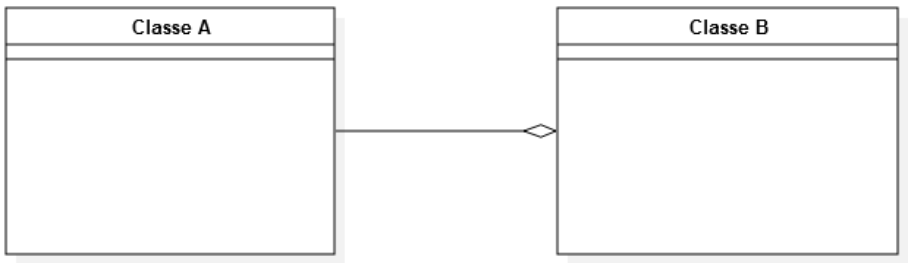

Figura 11: Diagramma delle classi - Aggregazione

- **Composizione**: simile all'aggregazione, ma indica una relazione più forte in cui l'esistenza delle parti dipende direttamente dall'oggetto principale. Una composizione è rappresentata da una linea con una freccia piena che punta dalla parte composta (parte) al tutto (oggetto principale). La linea può includere un rombo pieno sulla parte composta per indicare la composizione.

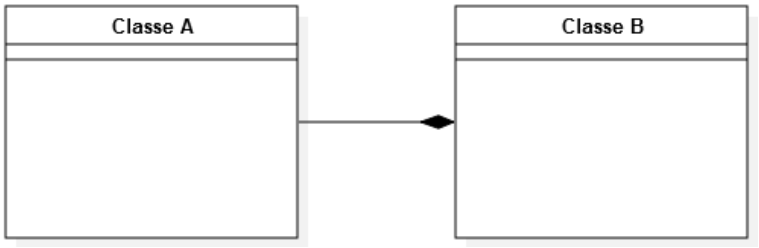

Figura 12: Diagramma delle classi - Composizione

- **Ereditarietà o generalizzazione**: indica una relazione di specializzazione tra una classe superiore (superclasse) e una o più classi inferiori (sottoclassi). Le sottoclassi ereditano gli attributi e i metodi della superclasse e possono estenderli o modificarli. Viene effigiata da una linea con una freccia che punta dalla sottoclasse alla superclasse. La linea può includere una freccia vuota con la scritta "extends" per indicare la generalizzazione.

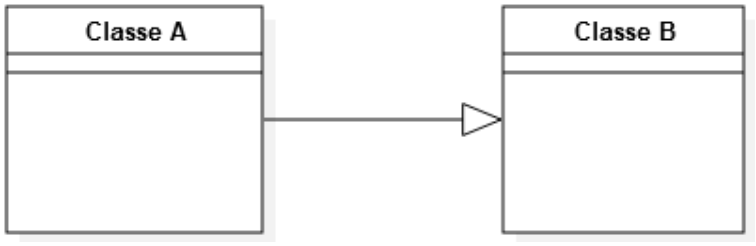

Figura 13: Diagramma delle classi - Generalizzazione

- **Dipendenza**: indica che una classe dipende dall'altra in un certo modo. Questo può significare che una classe utilizza o richiede i servizi di un'altra classe, ma non c'è un legame diretto tra le loro istanze. Una dipendenza è rappresentata da una linea tratteggiata che parte dalla classe dipendente e punta alla classe di cui dipende. La linea può includere una freccia vuota per indicare la direzione della dipendenza.

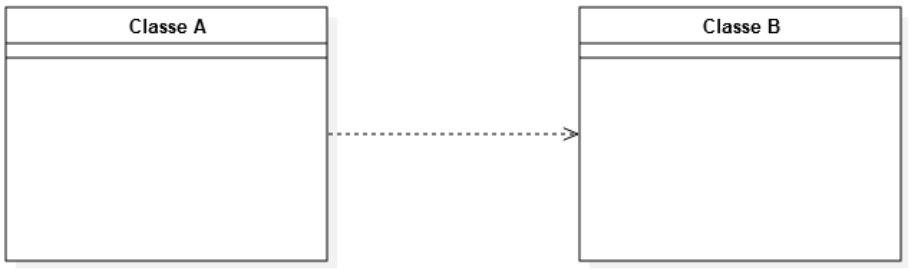

Figura 14: Diagramma delle classi - Dipendenza

- **Realizzazione**: indica che una classe implementa un'interfaccia o un contratto definito da un'altra classe. In altre parole, una classe realizza i metodi definiti da un'interfaccia o una classe astratta. Una realizzazione è raffigurata da una linea tratteggiata con una freccia vuota che punta dalla classe che implementa all'interfaccia o classe astratta che viene implementata.

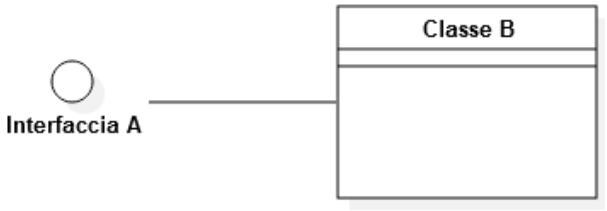

Figura 15: Diagramma delle classi - Realizzazione

2.2.3.6 Design pattern

I design pattern sono soluzioni progettuali generiche e riutilizzabili per problemi comuni che si presentano durante lo sviluppo del software. Sono modelli architetturali, concetti o paradigmi consolidati, spesso espressi attraverso codice o diagrammi, che forniscono una guida per risolvere specifiche problematiche di progettazione software in modo efficiente ed efficace. la documentazione sui design pattern serve come punto di riferimento centrale per tutti gli sviluppatori coinvolti nel progetto, consentendo loro di comprendere rapidamente e facilmente come sono stati implementati determinati pattern nel codice. Questo è particolarmente utile durante le fasi di progettazione, sviluppo e manutenzione del software, in quanto fornisce una chiara visione della struttura architetturale del sistema e dei pattern utilizzati.

2.2.3.7 Test

La parte dei test all'interno del processo di sviluppo software è un'attività critica volta a garantire che il prodotto finale soddisfi i requisiti funzionali, prestazionali e di qualità previsti. Questa fase coinvolge diverse attività, che includono la pianificazione dei test, la progettazione dei casi di test, l'esecuzione dei test e l'analisi dei risultati. In questa sezione *3.2.4* vengono fornite descrizioni dettagliate delle varie tipologie di test e della terminologia associata.

2.2.4 Codifica

2.2.4.1 Descrizione

La parte di codifica, nota anche come implementazione, rappresenta una fase critica e fondamentale nel processo di sviluppo del software. Durante questa fase, gli sviluppatori assumono il compito di tradurre i requisiti funzionali e non funzionali, delineati durante le fasi di analisi e progettazione, in codice eseguibile. Questo processo richiede

non solo competenze tecniche avanzate, ma anche una solida comprensione degli obiettivi del progetto e delle esigenze degli stakeholder.

2.2.4.2 Obiettivi

L'obiettivo principale della codifica è trasformare in modo accurato e efficiente i concetti astratti dei requisiti in istruzioni precise e comprensibili per il computer. Ciò implica la scrittura di codice pulito, ben strutturato e facilmente manutenibile, che sia in grado di soddisfare le esigenze funzionali del software e rispettare gli standard di qualità e le linee guida del progetto.

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;
- **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;
- **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;
- **gestione delle eccezioni**: il codice deve gestire correttamente le eccezioni e gli errori, fornendo messaggi di errore significativi e gestendo le situazioni anomale in modo appropriato;
- **riuso del codice**: il codice deve essere scritto in modo modulare e riutilizzabile, con la massima condivisione possibile di componenti e funzionalità tra i diversi moduli e classi;
- **test**: il codice deve essere testato e validato in modo approfondito, per garantire che soddisfi i requisiti funzionali e non funzionali e che sia privo di errori e bug;

- **sicurezza**: il codice deve essere scritto in modo sicuro, evitando vulnerabilità e falle di sicurezza, come l'iniezione di codice, la mancanza di controlli di input e la gestione non sicura delle password;
- **performance**: il codice deve essere ottimizzato per garantire prestazioni elevate e tempi di risposta rapidi, con l'uso corretto delle risorse e l'ottimizzazione dei cicli di elaborazione;
- **compatibilità**: il codice deve essere compatibile con le diverse piattaforme, sistemi operativi e browser, per garantire un'esperienza utente uniforme e coerente;
- **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 un ambiente di sviluppo integrato (IDE) sviluppato da Microsoft e adottato dal gruppo per lo sviluppo del software. Visual Studio Code offre funzionalità avanzate per la scrittura, la modifica e il debug del codice, con supporto per numerosi linguaggi di programmazione e framework.

2.2.4.5 Metriche

Metrica	Abbreviazione
23M-CC	Code Coverage
24M-BC	Branch Coverage
25M-SC	Statement Coverage

Tabella 2: Metriche riguardanti l'attività di codifica

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

L'utilizzo del file Docker garantisce che l'ambiente di sviluppo e produzione sia coerente e riproducibile. Questo approccio non solo assicura che le applicazioni funzionino correttamente in ogni fase del ciclo di vita, ma permette anche una gestione più efficiente delle dipendenze e delle configurazioni.
Di seguito un elenco delle best practice per la scrittura di file Docker:

- **chiarezza e semplicità**: sono principi fondamentali nella scrittura di un Dockerfile. Questi principi aiutano a creare immagini Docker che sono facili da comprendere, mantenere e utilizzare.;
- **versionamento**: è una pratica essenziale per mantenere il controllo sulle diverse versioni del software e delle dipendenze all'interno di un Dockerfile. Questa pratica aiuta a garantire la riproducibilità delle build, a facilitare il debug e a mantenere un ambiente di produzione stabile;
- **sicurezza**: la sicurezza è una priorità cruciale nella costruzione di immagini Docker. Un Dockerfile sicuro riduce la superficie di attacco, protegge dalle vulnerabilità e garantisce l'integrità dell'applicazione;
- **efficienza**: nell'ottica di garantire tempi di risposta bassi, mantenendo comunque le pretazioni, i file Docker devono avere un utilizzo corretto delle risorse e una gestione accurata dei container;
- **gestione delle variabili d'ambiente**: è importante stabilire una pratica rigorosa che renda chiara la configurazione dell'applicazione all'interno del container. Le variabili d'ambiente sono fondamentali per la configurazione dinamica dell'applicazione, consentendo di adattare il comportamento dell'applicazione senza modificare direttamente il codice sorgente;
- **monitoraggio**: riveste un'importanza fondamentale per garantire una corretta comprensione del comportamento dell'applicazione, individuare eventuali problemi e monitorare le prestazioni in tempo reale. Questo processo permette agli sviluppatori e agli amministratori di sistema di ottenere una visione dettagliata delle attività dell'applicazione, della sua efficienza e di eventuali criticità;

- **layering**: ogni istruzione nel Dockerfile crea un nuovo layer all'interno dell'immagine Docker. La gestione efficiente di questi strati è cruciale poiché influisce direttamente sulle dimensioni dell'immagine Docker finale e sulla velocità di build;
- **riduzione delle dimensioni delle immagini**: la riduzione delle dimensioni delle immagini Docker è un aspetto fondamentale per migliorare l'efficienza di deployment, ridurre i tempi di trasferimento e ottimizzare l'utilizzo delle risorse del sistema;
- **documentazione**: fornire una documentazione esaustiva e chiara con un Dockerfile è essenziale per garantire la comprensione del suo funzionamento, facilitare la manutenzione e favorire la collaborazione tra i membri del team;
- **testing**: testare un Dockerfile in modo completo è cruciale per garantire che l'immagine Docker risultante sia configurata correttamente e funzioni come previsto.

2.2.5.2 Strumenti

- **Docker**: è una piattaforma open-source che permette di creare, distribuire e gestire applicazioni in ambienti di sviluppo e produzione utilizzando container leggeri e autonomi;
- **Visual Studio Code**: è un ambiente di sviluppo integrato (IDE) sviluppato da Microsoft e adottato dal gruppo per lo sviluppo del software. Visual Studio Code offre funzionalità avanzate per la scrittura, la modifica e il debug del codice, con supporto per numerosi linguaggi di programmazione e framework.

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

Per la stesura della documentazione viene adottato l'approccio *Documentation as Code*, che consiste nel trattare la documentazione di progetto allo stesso modo del codice sorgente. Dunque vengono impiegati strumenti e metodologie tipiche dello sviluppo software:

- **versionamento**: la documentazione è versionata tramite il sistema di controllo di versione Git, in modo da tenere traccia delle modifiche e garantire la coerenza tra le varie versioni;
- **automazione**: l'automazione dei processi di build e di deploy permette di semplificare la gestione della documentazione e di ridurre il rischio di errori;

- **collaborazione**: l'uso di piattaforme di condivisione e collaborazione (GitHub) favorisce la cooperazione tra i membri del gruppo e semplifica la gestione dei documenti;
- **integrazione continua**: permette di verificare costantemente la correttezza e la coerenza della documentazione.

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 effettuare la scrittura in 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 responsabile della stesura e della cura del testo di un documento o di una sua parte in modo chiaro, preciso e accessibile. Per farlo, deve seguire lo stesso metodo utilizzato nello sviluppo del software, utilizzando il modello 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 nome_nuovo_Branch`

In particolare l'identificativo del branch deve essere 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, visibili a questo link. Una volta ultimata 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 nome_nuovo_Branch`
2. Se si riscontrano problemi nel precedente punto:
   - `git pull origin nome_nuovo_Branch`
3. Risolvere i conflitti e ripetere il punto 1.

Caso modifica documento in fase di redazione

Per continuare la redazione di un documento già in fase di stesura, sono necessari i seguenti comandi:

- `git pull`
- `git checkout nome_Branch`

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 Il resposabile

Il responsabile coordina e gestisce le attività del gruppo. In particolare:

- pianifica la documentazione;
- comunica con gli stakeholder;
- gestisce le risorse e gli strumenti.

3.1.5.3 L'amministratore

L'amministratore è colui che si occupa della gestione delle risorse e delle infrastrutture, inclusa la configurazione degli strumenti di supporto alla produzione del software. Inoltre, si occupa della gestione del corretto utilizzo delle procedure andando a garantire un'elevata efficienza e produttività del gruppo di lavoro.

3.1.5.4 I verificatori

Il ruolo e i doveri dei Verificatori sono esplicati in dettaglio al paragrafo 3.2.2.

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:

- titolo 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, rappresentato mediante una tabella. Il suo scopo è quello di fornire un tracciamento delle modifiche e aggiornamenti apportati al documento.La tabella riporta i seguenti dati:

- **versione** del documento;
- **data** di rilascio;
- nome del **Redattore**;
- 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. Tutte le figure e tabelle presenti in questo documento sono visibili negli appositi indici e sono direttamente accessibili tramite link ipertestuali.

3.1.6.4 Intestazione

Ogni pagina del documento, ad eccezione della prima, include un'intestazione che presenta, da sinistra a destra, il logo del gruppo, il titolo del documento e la versione.

3.1.6.5 Verbali: struttura generale

I verbali rappresentano un resoconto dettagliato degli incontri, tracciando gli argomenti discussi, le decisioni prese e le azioni da intraprendere. Sono suddivisi in verbali interni ed esterni, a seconda che il meeting coinvolga solo i membri del team o persone esterne al gruppo. La struttura è la medesima e prevede le seguenti sezioni:

- **prima pagina**:
  - tipologia verbale (interno / esterno);
  - titolo del documento;
  - data e versione del documento;

  - logo del gruppo
  - nome del gruppo.

- **Registro delle modifiche**;
- **informazioni sulla riunione**:
  - sede della riunione (interna / esterna);
  - orario di inizio e fine;
  - partecipanti del gruppo (interni);
  - partecipanti esterni.
- **Corpo del documento** All'interno del corpo del documento sono presenti le seguenti sezioni:
  - **ordine del giorno**: elenco di ciò che verrà discusso durante la riunione;
  - **sintesi dell'incontro**: breve riassunto delle discussioni e dei temi affrontati durante 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 delle attività;
    * assegnatari.
- **Ultima pagina**: presente solo in caso di verbale esterno, l'ultima pagina ha lo scopo di contenere data e firma di un componente esterno, presente durante l'incontro.

Il template dei verbali è disponibile nella repository del gruppo, più precisamente in questo link. (Ultima consultazione 2024-05-20).

3.1.7 Norme tipografiche

Nomi dei file
I nomi dei documenti devono essere coerenti con la loro tipologia, scritti in minuscolo e includere un riferimento alla versione del documento. In particolare, la denominazione dei file deve seguire la seguente 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 ";" mentre l'ultima prevede ".";
- i numeri razionali vengono scritti mediante l'uso della virgola come separatore tra parte intera e parte decimale;
- le date devono seguire lo standard internazionale ISO 8601, rappresentando la data con YYYY-MM-DD (anno, mese, giorno).
- le sezioni di un documento devono essere numerate in modo gerarchico, seguendo la struttura X.Y, dove:
  - X rappresenta il numero della sezione principale;
  - Y rappresenta il numero della sottosezione;

3.1.8 Abbreviazioni

Segue un elenco delle abbreviazioni più comuni utilizzate nei documenti:

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

Tabella 3: Spiegazione delle abbreviazioni utilizzate nei documenti.

3.1.9 Strumenti

Gli strumenti utilizzati per la redazione dei documenti sono:

- **LaTeX**: impiegato nella 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.

3.2 Verifica

3.2.1 Introduzione

Il processo di verifica è fondamentale durante tutto il ciclo di vita del software. Questo processo ha lo scopo di garantire che ogni attività sia corretta ed efficiente, identificando un processo di controllo per ogni prodotto realizzato. In particolare, ci si assicura che gli output del software (documentazione, codice sorgente, test, ecc.) siano conformi alle aspettative e ai requisiti specificati. Per fare ciò, è essenziale applicare tecniche e analisi di test seguendo procedure definite e adottando criteri affidabili. Le attività di verifica sono svolte dai Verificatori, i quali analizzano i prodotti e valutano 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. All'interno di esso vengono fornite 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. Questo processo coinvolge una serie di controlli sistematici e revisioni finalizzate a garantire che la documentazione sia accurata, completa e conforme agli standard e ai requisiti specificati. La verifica dei documenti assicura l'accuratezza, identificando e correggendo errori, omissioni e incongruenze. Questo riduce il rischio di malintesi o errori di implementazione, garantendo che tutte le informazioni tecniche e di progetto siano corrette e precise. Inoltre, la verifica garantisce che la documentazione rispetti gli standard aziendali, di settore e normativi, mantenendo la coerenza e la qualità attraverso tutti i documenti del progetto. Documenti verificati e accurati migliorano la comunicazione tra i membri del team e con gli stakeholder esterni, facilitando la comprensione e la collaborazione e riducendo i tempi di chiarimento e discussione. Essa si suddivide in:

- **correttezza tecnica**: vengono controllate le informazioni tecniche e i contenuti del documento, verificando che siano corretti e precisi;
- **conformità agli standard**: verifica che il documento segua le linee guida e gli standard stabiliti per la formattazione, la struttura e lo stile;

- **verifica della completezza**: si effettua un controllo per assicurare che tutti i requisiti documentali siano stati soddisfatti e che tutte le sezioni e le informazioni necessarie siano presenti;
- **revisione linguistica**: viene effettuata una correzione di eventuali errori grammaticali, ortografici e di punteggiatura;
- **coerenza**: ci si assicura che i termini e le definizioni siano utilizzati in modo uniforme e coerente in tutto il documento.

3.2.3 Analisi

L'analisi è un processo essenziale che mira a garantire la qualità e l'affidabilità del prodotto finale. Questa fase critica comporta un esame dettagliato e sistematico di tutte le componenti del progetto per identificare e risolvere eventuali problemi prima della consegna. Si suddivide in **analisi dinamica** e **analisi statica**.

3.2.3.1 Analisi dinamica

L'analisi dinamica riguarda l'esame del comportamento del software durante la sua esecuzione. Questo tipo di analisi viene eseguito in un ambiente di runtime e si concentra sull'osservazione di come il software interagisce con il sistema operativo, le risorse di sistema e altre applicazioni. L'analisi dinamica è essenziale per valutare l'efficienza, le prestazioni e la sicurezza del software in condizioni operative reali. I benefici dell'analisi dinamica includono:

- **individuazione di errori e difetti**: consente di identificare problemi e difetti nel software durante l'esecuzione, riducendo i rischi di malfunzionamenti e guasti;
- **ottimizzazione delle prestazioni**: permette di valutare le prestazioni del software e di identificare possibili aree di miglioramento;
- **convalida delle funzionalità**: assicura che il software funzioni correttamente e soddisfi i requisiti operativi attraverso test funzionali e di integrazione.

3.2.3.2 Analisi statica

L'analisi statica è il processo mediante il quale si esamina il codice sorgente di un programma senza eseguirlo. Questo tipo di analisi viene effettuato utilizzando strumenti

automatizzati che analizzano il codice per individuare possibili errori, violazioni delle best practice di codifica e potenziali vulnerabilità di sicurezza. L'analisi statica si concentra su aspetti come la sintassi, la semantica e le strutture di controllo del codice. I benefici di questo tipo di analisi sono molteplici, tra cui:

- **individuazione precoce degli errori**: consente di rilevare errori e difetti nelle prime fasi dello sviluppo, riducendo i costi e i tempi necessari per le correzioni;
- **miglioramento della qualità del codice**: promuove l'adozione di standard di codifica e best practice, migliorando la manutenibilità e la leggibilità del codice;
- **prevenzione di vulnerabilità di sicurezza**: aiuta a individuare vulnerabilità di sicurezza, come buffer overflow e injection flaws, prima che il software venga eseguito.

3.2.4 Testing

Il testing è una fase di molta importanza nel ciclo di vita dello sviluppo del software, il cui scopo primario è garantire che il prodotto finale sia di alta qualità, soddisfi i requisiti specificati e funzioni correttamente in tutte le condizioni previste. Questo processo si articola in una serie di attività sistematiche e metodiche per identificare difetti, errori e incongruenze nel software, migliorandone così l'affidabilità e le prestazioni complessive.

Per ogni test è necessario definire i seguenti aspetti:

- **identificazione del test**: un identificatore univoco per il caso di test, che faciliti il tracciamento e la referenza durante il processo di testing;
- **descrizione del test**: una descrizione chiara e concisa del caso di test, che spieghi la funzionalità specifica o il requisito che viene testato;
- **input del test**: dettagli sui dati di input necessari per eseguire il test, inclusi eventuali prerequisiti o condizioni iniziali;
- **output atteso**: descrizione degli output o dei risultati attesi del test, inclusi valori attesi, comportamenti del sistema o condizioni post-esecuzione;
- **procedura di esecuzione**: istruzioni dettagliate su come eseguire il test, comprese le azioni da compiere e le operazioni da eseguire per preparare l'ambiente di test e eseguire il test stesso;

- **criteri di accettazione**: criteri chiari e definiti per determinare se il test è stato superato con successo. Questi criteri dovrebbero essere basati sugli output attesi e sul comportamento desiderato del sistema;
- **passi di ripristino**: istruzioni su come ripristinare lo stato dell'ambiente di test o del sistema nel caso in cui il test abbia causato modifiche o danni;
- **risultato del test**: registrazione dei risultati del test, inclusi output, errori, problemi o comportamenti anomali riscontrati durante l'esecuzione del test;
- **stato del test**: indicazione dello stato del test, che può essere "superato", "non superato" o "non implementato";
- **commenti**: eventuali note aggiuntive.

3.2.4.1 Test di unità

I test di unità consentono di valutare il funzionamento delle singole unità o componenti del codice in modo isolato, senza dipendere dalle altre parti del sistema. Questo approccio mira a garantire che ogni porzione di software, anche la più piccola, operi correttamente e coerentemente con le specifiche definite, indipendentemente dalle interazioni con altre unità. L'obiettivo principale dei test di unità è di assicurare che ogni unità funzioni come previsto, restituendo risultati attesi per determinate condizioni di input. Ciò significa che, per ciascuna unità, vengono definiti uno o più scenari di test che includono input specifici e i risultati attesi per tali input. Durante l'esecuzione dei test, il comportamento effettivo dell'unità viene confrontato con i risultati attesi, e qualsiasi discrepanza tra i due indica un potenziale difetto nel codice. La pratica dei test di unità offre diversi vantaggi significativi. Innanzitutto, permette di individuare e risolvere tempestivamente eventuali difetti nel codice, poiché consente di identificare problemi già durante le prime fasi dello sviluppo, quando sono più facili e meno costosi da correggere. Inoltre, l'esecuzione automatica dei test di unità consente di garantire una verifica continua del codice, consentendo agli sviluppatori di individuare rapidamente eventuali regressioni o effetti collaterali indesiderati derivanti da modifiche al codice. Inoltre, i test di unità forniscono una documentazione vivente del comportamento del software, in quanto ciascun test rappresenta un caso specifico di utilizzo dell'unità. Questo non solo facilita la comprensione del codice da parte di altri sviluppatori, ma può anche servire come una sorta di contratto che definisce il comportamento atteso dell'unità nel tempo.

3.2.4.2 Test di integrazione

I test di integrazione sono una fase nel processo di testing del software durante la quale vengono verificati e validati i collegamenti e le interazioni tra le diverse unità o componenti del sistema. L'obiettivo principale dei test di integrazione è accertarsi che le singole parti del software, già testate individualmente durante i test di unità, funzionino correttamente quando integrate insieme come un sistema completo. In pratica, i test di integrazione esaminano come le varie unità o componenti interagiscono tra loro e se cooperano correttamente per fornire le funzionalità desiderate del software. Questi test possono rivelare problemi come:

- **incompatibilità tra le interfacce delle diverse unità**;
- **errori di comunicazione tra le componenti**;
- **mancata sincronizzazione o coerenza nei dati scambiati tra le unità**;

3.2.4.3 Test di sistema

Durante questa tipologia di test l'intero sistema viene valutato e verificato per garantire che soddisfi i requisiti funzionali e non funzionali specificati. Vengono eseguiti dopo i test di unità e di integrazione e sono progettati per valutare il sistema nel suo complesso, testando le sue funzionalità e le sue prestazioni rispetto agli obiettivi definiti. In particolare, permettono di:

- **verificare che il sistema soddisfi i requisiti specificati**;
- **valutare le prestazioni del sistema**;
- **verificare la sicurezza e la robustezza del sistema**;
- **verificare la compatibilità del sistema con l'ambiente di esecuzione**;
- **verificare la facilità d'uso e l'usabilità del sistema**.

3.2.4.4 Test di regressione

I test di regressione assicurano che le modifiche apportate al codice non abbiano introdotto nuovi difetti o rotto funzionalità esistenti nel software. Questi test vengono eseguiti dopo che sono state apportate modifiche al software, come nuove funzionalità,

correzioni di bug o aggiornamenti del sistema, per assicurare che il software funzioni correttamente anche dopo tali modifiche.

3.2.4.5 Test di accettazione

I test di accettazione sono una fase nel processo di sviluppo del software durante la quale il prodotto software viene testato per valutare se soddisfa i requisiti e le aspettative degli utenti finali, dei clienti o degli stakeholder. Questi test vengono eseguiti dopo che il software è stato completamente sviluppato e prima del suo rilascio ufficiale, consentendo agli utenti finali di valutare il software in un ambiente simile a quello di produzione e fornire feedback sulle sue funzionalità, l'usabilità e la conformità ai requisiti.

3.2.4.6 Sequenza delle fasi di test

I test vengono eseguiti in sequenza, partendo dai test di unità e procedendo verso i test di integrazione, di sistema, di regressione e di accettazione. Questo approccio graduale consente di identificare e risolvere i difetti in modo sistematico e strutturato, garantendo che il software funzioni correttamente e soddisfi i requisiti specificati.

3.2.4.7 Codici dei test

Ciascun test deve essere identificato da un codice univoco nel seguente formato:

[numero_test]-[tipologia]

Dove:

- **[numero_test]**: rappresenta un numero, in ordine crescente, associato al test, univoco all'interno della tipologia;
- **[tipologia]**: indica la tipologia di appartenenza del test, può assumere i seguenti valori:
  - **U**: test di unità;
  - **I**: test di integrazione;
  - **S**: test di sistema;
  - **R**: test di regressione;
  - **A**: test di accettazione.

3.2.4.8 Stato dei test

A ciascun test è associato uno stato che indica l'esito della sua esecuzione:

- **S**: il test è stato superato;
- **NS**: il test non è stato superato;
- **NI**: il test non è stato implementato.

Questi risultati saranno riportati nel documento *"Piano di Qualifica"*, in particolare nella sezione *"Metodologie di Testing"*.

3.3 Validazione

3.3.1 Introduzione

La validazione rappresenta un momento critico nel ciclo di sviluppo, in quanto sottopone il software a una serie di controlli dettagliati per assicurare la sua conformità ai requisiti stabiliti e la sua idoneità all'utilizzo da parte degli utenti finali. Questo processo non è solo una verifica formale, ma una fase cruciale che assicura che il software sia costruito in modo tale da rispondere pienamente alle esigenze e alle aspettative degli utenti, nonché agli obiettivi del progetto.

3.4 Gestione della configurazione

3.4.1 Introduzione

La gestione della configurazione è il processo di identificazione, controllo e coordinamento dei componenti software e delle risorse associate durante tutto il ciclo di vita del prodotto software. Questo processo assicura che il software e i suoi artefatti correlati siano gestiti in modo coerente e controllato, consentendo agli sviluppatori di tracciare le modifiche, gestire le versioni e garantire l'integrità e la coerenza del sistema nel tempo.

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:

- docs: contenente la documentazione prodotta;
- SyncCity: contenente il codice del progetto.

3.4.3.1 Struttura repository

Il repository inerente la documentazione è così organizzato:

1. **Candidatura**:
    - **verbali esterni**: al suo interno sono presenti i verbali delle riunioni avute con i membri di *SyncLab S.r.l.*;
    - **verbali interni**: contenente i verbali delle riunioni interne svolte;
    - **lettera di presentazione**: documento di presentazione per la candidatura al capitolato scelto;
    - **preventivo costi e assunzione impegni**: documento in cui vengono specificati i costi previsti, il totale delle ore per persona e gli impegni assunti;
    - **valutazione dei capitolati**: contenente il parere personale riguardo i capitolati offerti dalle varie aziende.
2. **RTB**:
    - **documentazione esterna**: contentente i seguenti documenti:
        - **Analisi dei Requisiti**;
        - **Piano di Progetto**;

  - **Piano di Qualifica**;
- **documentazione interna**: contentente i seguenti documenti:
  - **Glossario**;
  - **Norme di Progetto**.
- **verbali esterni**: al suo interno sono presenti i verbali delle riunioni avute con i membri di *SyncLab S.r.l.*;
- **verbali interni**: contenente i verbali delle riunioni interne svolte.

3. **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*. Questo workflow è un processo strutturato e collaborativo che consente ai membri del team di lavorare in modo efficace alla creazione, revisione e integrazione di documenti all'interno di un progetto. Questo approccio garantisce una gestione ordinata e controllata della redazione dei documenti, consentendo una migliore organizzazione e una maggiore qualità del lavoro finale.

Nomenclatura dei branch per le attività di redazione e/o modifica di documenti:

- il nome del nuovo branch deve riportare il titolo del documento da redarre o modificare;
- il nome dei verbali deve presentare anche la data della riunione:
  - *verbale_interno_yy_mm_dd* (es. verbale_interno_24_03_05);

3.4.4.2 Sviluppo

7Last utilizza Gitflow come flusso di lavoro.

Flusso di lavoro Gitflow:

1. **Branch develop**: è il punto di partenza per la creazione di nuovi branch, si crea a partire dal branch "main";

2. **Branch release**: è un ramo del repository Git dedicato alla preparazione e al rilascio di una nuova versione del software o del progetto;
3. **Branch feature**: utilizzato per lo sviluppo di nuove funzionalità o per l'implementazione di miglioramenti specifici all'interno di un progetto software;
4. **Merge di feature in develop**: consente di integrare le nuove funzionalità sviluppate in un ambiente isolato nel branch di sviluppo principale del progetto;
5. **Merge di release in develop e main**: consente di integrare le modifiche e le correzioni effettuate durante la preparazione di una release nel flusso di lavoro principale del progetto;
6. **Branch hotfix**: utilizzato per affrontare correzioni di bug critici o problemi urgenti che richiedono una risoluzione immediata e non possono attendere il normale ciclo di rilascio del software;
7. **Merge di hotfix in develop e main**: utilizzato per integrare le correzioni di bug critici o problemi urgenti nei flussi di lavoro principali del progetto.

3.4.4.3 Pull Request

Una pull request è una richiesta che un membro del team di sviluppo fa al Verificatore attuale del team per revisionare e, eventualmente, accettare le modifiche apportate a un repository Git. Questo processo è essenziale per la collaborazione e la gestione del codice all'interno di un progetto software. Ecco una descrizione dettagliata di una pull request:

1. creazione della pull request: dopo aver completato le modifiche nel suo branch locale, questo membro crea una pull request per richiedere che le sue modifiche siano integrate nel branch principale del repository;
2. selezionare il branch di partenza e quello di destinazione;
3. cliccare su "Create Pull Request";
4. assegnare un titolo alla Pull Request;
5. aggiungere una descrizione delle modifiche apportate;
6. Assegnare un Verificatore;

7. Per convenzione, l'assegnatario è colui che richiede la Pull Request;
8. Selezionare le label;
9. Selezionare il progetto;
10. Selezionare la milestone;
11. Cliccare su "Create Pull Request";
12. 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

Una Change Request (richiesta di modifica) è una proposta formale per modificare uno o più aspetti del progetto. Questo può includere modifiche al codice sorgente, alla documentazione, alle specifiche, ai requisiti, alle interfacce utente, o qualsiasi altro elemento del progetto. Per lo svolgimento di questo processo in modo ordinato e strutturato seguiamo lo standard ISO/IEC 12207:1995, il quale prevede le seguenti attività:

1. **identificazione e registrazione**: identificare e documentare in modo univoco tutti gli elementi di configurazione del software. Questo include il codice sorgente, i file binari, la documentazione, le specifiche, i piani di test e qualsiasi altro artefatto correlato. Ogni elemento di configurazione deve avere un identificatore unico e deve essere registrato in un database o un sistema di gestione delle configurazioni per garantire la tracciabilità e il controllo;
2. **valutazione e analisi**: valutare le richieste di modifica per determinare il loro impatto sul sistema e sugli altri elementi di configurazione. Questa fase prevede un'analisi tecnica dettagliata per identificare i potenziali rischi, benefici, costi e tempi associati alla modifica proposta. Si considerano anche le dipendenze tra i vari elementi di configurazione per evitare effetti indesiderati;
3. **approvazione o rifiuto**: decidere se accettare o rifiutare le richieste di modifica basandosi sull'analisi e valutazione effettuate. Questo processo coinvolge spesso un comitato di controllo delle modifiche (Change Control Board - CCB) che esamina le richieste, discute le implicazioni e prende decisioni formali. L'approvazione o il rifiuto devono essere documentati ufficialmente;

4. **pianificazione delle modifiche**: pianificare la realizzazione delle modifiche approvate, definendo le attività necessarie, le risorse richieste, i tempi di esecuzione e i responsabili. La pianificazione deve essere dettagliata e deve includere tutte le fasi della modifica, dalla progettazione e sviluppo fino alla verifica e rilascio. È importante anche pianificare le strategie di rollback nel caso in cui la modifica non abbia successo;

5. **implementazione**: realizzare le modifiche approvate nel sistema di configurazione, seguendo il piano stabilito. Questa fase comprende la codifica, la modifica dei documenti, la preparazione dei pacchetti di rilascio e l'aggiornamento del sistema di gestione delle configurazioni con le nuove versioni degli elementi. L'implementazione deve essere eseguita in modo controllato per garantire la qualità e l'integrità del software;

6. **verifica e validazione**: verificare e validare le modifiche implementate per assicurarsi che soddisfino i requisiti specificati e che non abbiano introdotto nuovi difetti. Questa attività include la revisione del codice, l'esecuzione di test, la valutazione delle prestazioni e l'ispezione della documentazione. La verifica e la validazione garantiscono che le modifiche siano corrette e che il sistema funzioni come previsto;

7. **documentazione**: documentare tutte le attività di controllo di configurazione, incluse le richieste di modifica, le decisioni di approvazione o rifiuto, i piani di modifica, le attività di implementazione e i risultati delle verifiche. La documentazione deve essere completa e accurata per mantenere la tracciabilità e per supportare eventuali audit futuri. Tutti i documenti devono essere aggiornati e conservati in un sistema di gestione delle configurazioni;

8. **comunicazione agli stakeholder**: informare tutti gli stakeholder rilevanti riguardo alle modifiche effettuate, allo stato delle configurazioni e alle decisioni prese. La comunicazione deve essere chiara e tempestiva, utilizzando canali appropriati come riunioni, report, email o sistemi di gestione dei progetti.

3.4.6 Release management and delivery

Il processo di "Release Management and Delivery" secondo lo standard ISO/IEC 12207:1995 è una componente essenziale del ciclo di vita del software, che si concentra sulla gestione delle versioni del software e sulla distribuzione dei prodotti software agli utenti finali. Questo processo assicura che le versioni del software siano rilasciate in modo controllato, pianificato e che soddisfino i requisiti di qualità e sicurezza. Attraverso una serie di attività ben definite, il processo di gestione del rilascio garantisce che il software sia pronto per l'uso, minimizzando i rischi di problemi post-rilascio e massimizzando l'efficienza operativa. La creazione della prima release deve avvenire in concomitanza con la baseline RTB (Requirements and Technology Baseline), mentre la creazione delle successive release deve avvenire in concomitanza con la baseline PB (Product Baseline) e, se prevista, CA (Customer Acceptance).

3.4.6.1 Procedura per la creazione di una release

1. Accedere alla pagina del repository;
2. nella pagina principale del repository, clicca sulla scheda "Releases" situata nella barra di navigazione in alto;
3. clicca sul pulsante "Draft a new release" o "Create a new release" se non ci sono release precedenti;
4. inserisci un tag per la release. Se il tag non esiste, GitHub lo creerà per te. Il tag dovrebbe seguire una convenzione di versioning;
5. inserisci un titolo e una descrizione significativi per la release;
6. seleziona "Publish release" per rendere la release disponibile pubblicamente.

Seguendo questi passaggi, il gruppo potrà creare e pubblicare una release ben strutturata e informativa su GitHub, garantendo che tutti gli utenti abbiano accesso alle ultime funzionalità e miglioramenti del software.

3.5 Joint review

3.5.1 Introduzione

La Joint Review è un'attività collaborativa cruciale in cui i recensori e i recensiti del progetto si incontrano per esaminare e valutare diversi aspetti del progetto. Nel nostro

caso i recensori sono costituiti da proponente, committente e stakeholder; mentre i recensiti sono rappresentati da noi. Lo scopo principale di questa revisione congiunta è garantire che tutte le parti coinvolte abbiano una comprensione comune dello stato attuale del progetto e dei passi successivi necessari per raggiungere gli obiettivi.

3.5.2 Implementazione del processo

L'implementazione del processo comprende i seguenti impegni:

3.5.2.1 Revisioni periodiche

In corrispondenza delle milestone stabilite saranno effettuate delle revisioni periodiche, come riportato nel documento *Piano di Progetto*.

3.5.2.2 Stato Avanzamento Lavori

Al termine di ogni sprint viene svolta una revisione SAL (Stato Avanzamento Lavori) tra il team e il proponente. Questa revisione ha lo scopo di valutare il lavoro svolto durante lo sprint precedente, per verificare che gli obiettivi prefissati siano stati correttamente raggiunti secondo le scadenze prefissate. Inoltre, durante questo incontro, si pianificano le attività per lo sprint successivo.

3.5.2.3 Revisioni ad hoc

Se uno qualsiasi dei soggetti coinvolti tra gli stakeholder lo ritenga opportuno, è possibile istituire una revisione ad hoc per esaminare attentamente lo stato di avanzamento dei lavori. Durante questa revisione, si discutono eventuali problematiche emerse e le relative soluzioni adottabili.

3.5.2.4 Risorse per le revisioni

Le risorse necessarie per condurre le revisioni possono essere di varia natura, comprendendo personale, strumenti, hardware, software, strutture, e così via. È fondamentale che tali risorse siano discusse e concordate tra tutte le parti coinvolte.

3.5.2.5 Elementi da concordare

Pochi giorni prima della riunione con la proponente *7Last* si impegna a consegnare un report con tutte le modifiche effettuate e i campi in cui sono state applicate. In ciascuna revisione rimane la necessità di concordare i seguenti elementi:

- agenda della riunione;
- prodotti software risultati dall'attività e relative problematiche;

3.5.2.6 Documenti e distribuzione dei risultati

Tramite i *Verbali Esterni* vengono documentati i risultati delle revisioni, compresi i problemi individuati, le soluzioni proposte e le azioni correttive da intraprendere. Questi documenti vengono distribuiti a tutte le parti coinvolte per garantire una comprensione comune e una chiara comunicazione. La parte recensente comunicherà alla parte recensita la veridicità di quanto riportato, approvando o disapprovando i documenti citati.

3.6 Risoluzione dei problemi

3.6.1 Introduzione

La risoluzione dei problemi è un processo che mira a identificare, analizzare e risolvere le varie problematiche che possono emergere durante lo sviluppo. Si riferisce a un insieme di tecniche e metodologie utilizzate per affrontare e risolvere le difficoltà tecniche, di design, di implementazione o di gestione che possono sorgere durante lo sviluppo del software. Questo processo può includere:

- **identificazione del problema**: consiste nel riconoscere e definire chiaramente il problema;
- **analisi del problema**: nella quale si esamina il problema per capire le cause sottostanti e l'impatto;
- **generazione di soluzioni**: vengono proposte diverse possibili soluzioni al problema;
- **valutazione delle soluzioni**: vengono analizzate le soluzioni proposte per determinarne la fattibilità, l'efficacia e l'efficienza;

- **implementazione della soluzione**: la soluzione adottata viene implementata e testata per verificare che risolva il problema in modo efficace.

3.6.2 Gestione dei rischi

All'interno del documento *Piano di Progetto*, più precisamente nella sezione *Analisi dei rischi* sono contenuti i rischi che potrebbero emergere durante lo svolgimento del progetto, con relativi approfondimenti e strategie di mitigazione.

3.6.2.1 Codifica dei rischi

Ogni rischio è identificato da un codice univoco nel seguente formato:

R [tipologia] [indice]: nome identificativo del rischio

Dove:

- **[tipologia]**: rappresenta la categoria di rischio, la quale può essere organizzativa, tecnologica o comunicativa;
- **[indice]**: un valore numerico incrementale che identifica univocamente il rischio per ogni tipologia: un rischio elevato equivale a 3, un rischio medio equivale a 2, mentre un rischio basso equivale a 1.

3.6.2.2 Metriche

Metrica	Nome
29M-NCR	Rischi non calcolati

Tabella 4: Metriche relative alla gestione dei processi

3.6.3 Strumenti

- **ClickUp**: utilizzato per la gestione delle attività e delle issue;

3.7 Gestione della qualità

3.7.1 Introduzione

La gestione della qualità consiste in un insieme di attività e processi volti a garantire che il software sviluppato soddisfi gli standard di qualità richiesti. Questo include l'adesione a requisiti specificati, normative, e le aspettative degli stakeholder. La gestione della qualità è cruciale per assicurare che il prodotto finale sia affidabile, sicuro, e performante.

3.7.2 Attività

Il processo di gestione della qualità comprende le seguenti attività:

1. **pianificazione della qualità**: vengono definiti gli standard di qualità e le metriche che il progetto deve raggiungere;
2. **assicurazione della qualità**: consiste in attività sistematiche e pianificate per garantire che i processi utilizzati durante lo sviluppo del software siano adeguati e seguiti correttamente;
3. **controllo della qualità**: consiste in ispezioni, test e revisioni tecniche per verificare che il prodotto soddisfi gli standard di qualità definiti;
4. **gestione dei requisiti**: assicurarsi che i requisiti siano chiari, completi e testabili, e che il software sviluppato soddisfi tali requisiti. Inoltre è importante tenere la tracciabilità tra i requisiti e le varie fasi del ciclo di vita del software per garantire che tutte le richieste siano soddisfatte;
5. **gestione delle configurazioni**: attraverso un sistema di gestione delle configurazioni ci si assicura che ciascuna di esse venga sottoposta ad opportuni controlli e valutazioni;
6. **monitoraggio e reporting**: in questa fase vengono generati report che descrivono lo stato della qualità del progetto, inclusi i risultati dei test, i difetti trovati e le azioni correttive intraprese;
7. **miglioramento continuo**: applicare le modifiche ai processi basate sui feedback e sulle analisi per migliorare continuamente la qualità del software e dei processi di sviluppo.

3.7.3 Piano di qualifica

Il documento *Piano di Qualifica* è progettato per garantire che il software sviluppato rispetti gli standard di qualità richiesti e soddisfi le aspettative degli stakeholder. Il suo utilizzo e il suo scopo si estendono a diverse aree critiche del progetto, dalle fasi iniziali di pianificazione fino alla consegna finale del prodotto.

3.7.4 Ciclo di Deming

Il *ciclo di Deming*, è un ciclo a 4 stadi che consente di apportare specifiche migliorie a specifici processi. È composto da:

- **Plan**: in cui si definiscono le attività, scadenze, responsabilità e risorse per raggiungere specifici obiettivi di miglioramento;
- **Do**: esecuzione delle attività definite dal punto precedente;
- **Check**: verifica l'esito delle azioni di miglioramento rispetto alle attese;
- **Act**: consolidare il buono e cercare modi per migliorare il resto.

Riteniamo di fondamentale importanza specificare che questo ciclo **non opera** sul prodotto, bensì sul ***way of working*** del gruppo, per migliorarlo e renderlo più efficiente.

3.7.5 Struttura e identificazioni metriche

Ogni metrica presenta la seguente struttura:

- **Metrica**: codice identificativo nel formato:

  [numero]M-[acronimo]

  Dove:

  - **[numero]**: numero progressivo univoco per ogni metrica;
  - **M**: metrica;
  - **[acronimo]**: abbreviazione del nome della metrica.

- **Nome**: nome della metrica;
- **Valore accettabile**: valore minimo affinchè la metrica sia considerabile soddisfacente e conforme agli obiettivi di qualità;

- **Valore ammissibile**: valore ottimale e ideale che dovrebbe essere raggiunto dalla metrica;
- **Valore ottimo**:
- **Descrizione**: breve descrizione della metrica adottata e delle sue funzionalità;

3.7.6 Metriche

Metrica	Nome
30M-QMS	Metriche di qualità soddisfatte

Tabella 5: Metriche relative alla gestione della qualità

4 Processi organizzativi

I processi organizzativi sono fondamentali per garantire che il progetto sia gestito in modo efficace, efficiente e conforme agli standard di qualità. Questi processi servono a coordinare le attività del team, a ottimizzare l'uso delle risorse e a mitigare i rischi, assicurando così il successo del progetto.

4.1 Gestione dei processi

4.1.1 Introduzione

Le attività di gestione dei processi sono:

- **pianificazione dei processi**: vengono definiti gli obiettivi, le fasi del progetto, le risorse necessarie e le scadenze. Questa fase stabilisce anche i criteri di successo e il piano di lavoro dettagliato;
- **definizione dei processi**: documentazione dei processi chiave che saranno utilizzati nel progetto, inclusi i processi di sviluppo del software, di controllo di versione, di gestione dei cambiamenti e di assicurazione della qualità;
- **assegnazione delle risorse**: assegnazione dei membri del team alle attività specifiche del progetto in base alle loro competenze e disponibilità;
- **monitoraggio e controllo**: si effettua un monitoraggio continuo del progresso del progetto rispetto al piano stabilito. Questo include il monitoraggio dei tempi, dei costi e della qualità, nonché l'identificazione e la gestione dei rischi;
- **gestione dei cambiamenti**: vengono effettuate una valutazione e gestione delle modifiche richieste durante lo sviluppo del software. Questo può includere modifiche ai requisiti, alla pianificazione o alla distribuzione delle risorse;
- **assicurazione della qualità**: implementazione di processi e procedure per garantire che il prodotto software soddisfi i requisiti e le aspettative del cliente;
- **comunicazione e coordinamento**: facilitazione della comunicazione tra i membri del team, gli stakeholder e altri soggetti interessati al progetto. Questo assicura che tutte le parti coinvolte siano informate sullo stato del progetto e sulle decisioni prese;

- **miglioramento continuo**: consiste in un'analisi dei processi utilizzati nel progetto al fine di identificare aree di miglioramento e implementare azioni correttive per ottimizzare l'efficienza e la qualità complessiva del lavoro svolto.

4.1.2 Pianificazione

4.1.2.1 Descrizione

La pianificazione dei processi implica la definizione, l'organizzazione e il controllo delle attività necessarie per portare a termine con successo il progetto. Si tratta di un'attività strategica che fornisce una guida chiara e una struttura gestionale per tutto il ciclo di vita del progetto. La pianificazione dei processi è essenziale per garantire che il progetto sia completato in tempo, entro il budget e con la qualità richiesta.

4.1.2.2 Obiettivi

Lo scopo principale della pianificazione dei processi è garantire che il progetto venga eseguito in modo efficiente, efficace e conforme agli obiettivi e ai requisiti stabiliti, assicurando allo stesso tempo che ciascun membro del team assuma ogni ruolo per almeno una volta. Questo processo serve anche a mitigare i rischi e ad affrontare le sfide in modo proattivo, consentendo al team di affrontare eventuali ostacoli lungo il percorso.

4.1.2.3 Assegnazione dei ruoli

Durante lo svolgimento del progetto, i membri di *7Last* assumeranno ruoli distinti, tra cui:

- **responsabile**, il quale si occupa di:
  - coordinare il gruppo di lavoro;
  - pianificare e controllare le attività;
  - gestire le risorse;
  - gestire le comunicazioni con l'esterno.
- **Amministratore**, i suoi compiti sono:
  - gestire l'ambiente di lavoro;
  - gestione delle procedure e delle norme;

  - gestione della configurazione del prodotto.

- **Analista**, che:
  - analizza i requisiti del progetto;
  - redige l'analisi dei requisiti;
  - studia il dominio applicativo del problema.
- **Progettista**, il quale:
  - progetta l'architettura del prodotto;
  - prende decisioni tecniche e tecnologiche.
- **Programmatore**: si occupa di:
  - scrivere il codice del prodotto;
  - implementare le funzionalità richieste;
  - codifica le componenti dell'architettura del prodotto;
  - redige il *Manuale Utente*.
- **Verificatore**: il cui compito è:
  - verificare che il lavoro svolto sia conforme alle norme e alle specifiche tecniche del progetto;
  - ricercare ed eventualmente segnalare errori;
  - redigere la sezione retrospettiva del *Piano di Qualifica*.

4.1.2.4 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

La fase di coordinamento rappresenta il momento in cui convergono le molteplici variabili coinvolte nell'esecuzione del lavoro. Questa fase è come il nodo centrale di un intricato intreccio, in cui le attività, le risorse e le comunicazioni si fondono per creare un ambiente che facilita un'esecuzione efficace del progetto. È il punto di congiunzione vitale in cui le diverse componenti del progetto si uniscono, fornendo un quadro organizzativo che funge da guida e sostegno per il raggiungimento degli obiettivi prefissati.

4.1.3.2 Obiettivi

Lo scopo principale della fase di coordinamento è quello di creare armonia tra tutte le attività del progetto, integrando e sincronizzando ogni componente in modo da massimizzare l'efficienza e ottimizzare l'utilizzo delle risorse disponibili. Questo processo implica una gestione attenta delle risorse umane, finanziarie e temporali, assicurando che ogni membro del team sia adeguatamente informato, motivato e allineato sui compiti da svolgere e sugli obiettivi da raggiungere. Inoltre, il coordinamento mira a garantire che le risorse siano allocate in modo appropriato, evitando sovrapposizioni o carenze che potrebbero compromettere il successo del progetto. In definitiva, la fase di coordinamento agisce come il collante che tiene insieme tutte le componenti del progetto, consentendo al team di lavorare in modo collaborativo verso il raggiungimento del successo.

A tal proposito, il gruppo *7Last* si occupa di mantenere comunicazioni attive, sia interne che esterne, che possono essere sincrone o asincrone.

4.1.3.3 Comunicazioni asincrone

- **comunicazione asincrone interne**: viene deciso di adottare Telegram, applicazione che consente di comunicare mediante l'utilizzo di messaggi testo, media e file in chat private o all'interno di gruppi;
- **comunicazione asincrone esterne**: vengono adottati due canali differenti per garantire le comunicazioni asincrone esterne, ovvero:

- **E-mail**: per comunicazioni formali e ufficiali;
- **Discord**: in caso di necessità di risposta immediata e per comunicazioni informali.

4.1.3.4 Comunicazioni sincrone

- **comunicazione sincrone interne**: viene adottato Discord per questo scopo, il quale permette di comunicare tramite chiamate vocali, videochiamate, messaggi di testo, media e file in chat private o all'interno di gruppi;
- **comunicazione sincrone esterne**: in accordo con l'azienda *SyncLab S.r.l.* viene scelto di adottare Google Meet per le comunicazioni sincrone esterne.

4.1.3.5 Riunioni interne

Le riunioni interne avranno luogo ogni mercoledì, tramite Discord. Queste riunioni serviranno a monitorare il progresso delle attività, a discutere eventuali problematiche riscontrate e a pianificare le attività future. L'orario in cui si terranno queste riunioni è dalle 15:00 alle 16:00. Nel caso in cui un membro del gruppo non possa partecipare alla riunione, è tenuto a comunicarlo tempestivamente al responsabile, il quale provvederà a trovare un'altra data e un altro orario che possa andare bene a tutti i membri del gruppo. Le riunioni interne saranno registrate e il verbale sarà redatto dal redattore della riunione. In queste riunioni i compiti del responsabile sono:

- stabilire l'***ordine del giorno*** della riunione;
- guidare la riunione e assicurarsi che tutti i membri forniscano il loro parere in modo ordinato;
- pianificare e assegnare le nuove attività da svolgere.

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:

7Last si impegna ad organizzare incontri regolari con *SyncLab S.r.l.* per monitorare lo stato di avanzamento del progetto e affrontare eventuali dubbi o questioni. In linea con tale impegno, è stato concordato di tenere incontri di Stato Avanzamento Lavori (SAL) inizialmente ogni due settimane, con l'intenzione di aumentare la frequenza a uno ogni settimana in seguito. Gli incontri si svolgeranno tramite la piattaforma Google Meet e tratteranno i seguenti argomenti:

- discussione e valutazione delle attività svolte nel periodo passato;
- pianificazione delle attività per il periodo successivo;
- chiarimento di eventuali dubbi emersi nel corso del periodo passato.

Verbali esterni

Come avviene per le riunioni interne, anche per quelle esterne verrà stilato un verbale secondo le medesime modalità illustrate nella sezione relativa ai Verbali Interni. Tuttavia, la distinzione risiede nel fatto che il verbale redatto sarà successivamente inviato all'azienda per l'approvazione e la firma.

4.1.3.7 Strumenti

- **Discord**: impiegato per la comunicazione sincrona e i meeting interni del team;
- **Telegram**: utilizzato per la comunicazione asincrona interna;
- **Google Meet**: adottato per le comunicazioni sincrone esterne;
- **Gmail**: come servizio di posta elettronica.

4.1.3.8 Metriche

Metrica	Nome
33M-BV	Budget Variance
32M-SV	Schedule Variance

Tabella 6: Metriche relative alla gestione della qualità

4.2 Miglioramento

4.2.1 Introduzione

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

- **analisi dei dati**: questa fase coinvolge la raccolta e l'analisi dei dati pertinenti per identificare le aree che richiedono miglioramento;
- **valutazione delle prestazioni attuali**: consiste nel valutare le prestazioni attuali rispetto agli obiettivi desiderati o agli standard di riferimento. Questo aiuta a determinare dove sono necessari miglioramenti e quali sono le aree critiche che richiedono attenzione;
- **identificazione delle aree di miglioramento**: basandosi sull'analisi dei dati e sulla valutazione delle prestazioni attuali, vengono identificate le specifiche aree o processi che richiedono miglioramento;
- **pianificazione del miglioramento**: una volta identificate le aree di miglioramento, viene sviluppato un piano dettagliato che definisce gli obiettivi del miglioramento, le attività necessarie, le risorse coinvolte, le tempistiche e le metriche per valutare il successo;
- **implementazione delle modifiche**: coinvolge l'attuazione del piano di miglioramento, che può includere l'introduzione di nuovi processi, procedure o strumenti, la formazione del personale, la revisione dei flussi di lavoro esistenti, ecc..;

- **monitoraggio e valutazione**: una volta implementate le modifiche, è necessario monitorare e valutare i risultati per assicurarsi che il miglioramento sia stato efficace e che i risultati attesi siano stati raggiunti;
- **miglioramento**: attuazione di azioni correttive e preventive per ottimizzare i processi.

4.3 Formazione

4.3.1 Introduzione

La formazione è una componente essenziale per garantire che tutti i membri del team siano adeguatamente preparati per affrontare le sfide tecniche e gestionali del progetto. Questa fase del progetto si concentra sullo sviluppo delle competenze e delle conoscenze necessarie per utilizzare strumenti, tecnologie e metodologie specifiche del progetto, promuovendo così l'efficienza e la qualità del lavoro svolto.

4.3.2 Metodo di formazione

4.3.2.1 Individuale

Ogni inviduo del team dovrà compiere un processo di autoformazione per riuscire a svolgere il ruolo assegnato al meglio. La rotazione dei ruoli permetterà al nuovo occupante di un ruolo di apprendere le competenze necessarie da colui che ha precedentemente svolto il ruolo, nel caso avesse delle lacune. Questo metodo permette di avere una formazione continua e di garantire che ogni membro del team sia in grado di svolgere ogni ruolo.

4.3.2.2 Gruppo

SyncLab S.r.l. mette a disposizione sessioni formative riguardo le tecnologie adottate nel progetto.

5 Standard per la qualità

Abbiamo scelto di adottare standard internazionali per l'analisi e la valutazione della qualità dei processi e del software. In particolare, la suddivisione dei processi in primari, di supporto e organizzativi sarà guidata dall'adozione dello standard ISO/IEC 12207:1995. Inoltre, l'adozione dello standard ISO/IEC 25010:2023 ci fornirà un quadro completo per la definizione e la classificazione delle metriche di qualità del software. Abbiamo deciso di applicare solo questi due standard, poiché lo standard ISO/IEC 9126:2001 è stato ritirato e sostituito dallo standard ISO/IEC 25010:2023.

5.1 Caratteristiche del sistema ISO/IEC 25010:2023

5.1.1 Appropriatezza funzionale

- **Completezza**: il prodotto software deve soddisfare tutti i requisiti definiti e attesi dagli utenti;
- **Correttezza**: il prodotto software deve funzionare come previsto e produrre risultati accurati;
- **Appropriatezza**: il prodotto software deve essere adatto allo scopo previsto e al contesto di utilizzo.

5.1.2 Performance

- **Tempo**: il prodotto software deve rispettare le scadenze e i tempi di consegna previsti;
- **Risorse**: il prodotto software deve utilizzare le risorse di sistema in modo efficiente e ragionevole;
- **Capacità**: il prodotto software deve essere in grado di gestire il carico di lavoro previsto.

5.1.3 Compatibilità

- **Coesistenza**: il prodotto software deve essere in grado di coesistere con altri software e sistemi sul computer;

- **Interoperabilità**: il prodotto software deve essere in grado di scambiare informazioni e collaborare con altri software e sistemi.

5.1.4 Usabilità

- **Riconoscibilità**: il prodotto software deve avere un'interfaccia utente intuitiva e facile da usare;
- **Apprendibilità**: gli utenti devono essere in grado di imparare a utilizzare il prodotto software in modo rapido e semplice;
- **Operabilità**: il prodotto software deve essere facile da usare e da controllare;
- **Protezione** errori: il prodotto software deve essere in grado di rilevare e gestire gli errori in modo efficace;
- **Esteticità**: il prodotto software deve avere un'interfaccia utente piacevole e accattivante;
- **Accessibilità**: il prodotto software deve essere accessibile a persone con disabilità.

5.1.5 Affidabilità

- **Maturità**: il prodotto software deve essere stabile, affidabile e robusto;
- **Disponibilità**: il prodotto software deve essere disponibile quando necessario;
- **Tolleranza**: il prodotto software deve essere in grado di tollerare errori e condizioni inaspettate;
- **Ricoverabilità**: il prodotto software deve essere in grado di ripristinare i dati e le funzionalità in caso di guasto o errore.

5.1.6 Sicurezza

- **Riservatezza**: il prodotto software deve proteggere i dati sensibili e le informazioni riservate;
- **Integrità**: il prodotto software deve garantire l'accuratezza e la completezza dei dati;

- **Non ripudio**: il prodotto software deve garantire che le transazioni e le comunicazioni non possano essere negate o ripudiate;
- **Autenticazione**: il prodotto software deve verificare l'identità degli utenti e garantire che solo gli utenti autorizzati possano accedere al sistema;
- **Autenticità**: il prodotto software deve garantire che l'origine dei dati e delle informazioni sia verificabile.

5.1.7 Manutenibilità

- **Modularità**: il prodotto software deve essere progettato in modo modulare, con componenti indipendenti e ben definiti;
- **Riusabilità**: i componenti del prodotto software devono essere progettati per essere riutilizzati in altri progetti;
- **Analizzabilità**: il prodotto software deve essere progettato in modo da essere facilmente analizzabile e comprensibile;
- **Modificabilità**: il prodotto software deve essere progettato in modo da essere facilmente modificabile e adattabile;
- **Testabilità**: il prodotto software deve essere progettato in modo da essere facilmente testabile.

5.1.8 Portabilità

- **Adattabilità**: il prodotto software deve essere in grado di adattarsi a nuovi ambienti, requisiti e tecnologie;
- **Installabilità**: il prodotto software deve essere facilmente installabile e configurabile;
- **Sostituibilità**: il prodotto software deve essere facilmente sostituibile con altre soluzioni o versioni più recenti.

5.2 Suddivisione secondo standard ISO/IEC 12207:1995

5.2.1 Processi primari

Essenziali per lo sviluppo del software e comprendono:

- **acquisizione**: gestione dei propri sotto-fornitori;
- **fornitura**: gestione delle relazioni con il cliente;
- **sviluppo**: comprende tutte le attività legate alla progettazione, implementazione e verifica del software;
- **operazione**: installazione ed fornitura dei prodotti e/o servizi;
- **manutenzione**: correzione, adattamento, progressione.

5.2.2 Processi di supporto

Questi processi forniscono il supporto necessario per i processi primari e comprendono:

- **documentazione**: comprende la generazione e la cura della documentazione correlata al software;
- **gestione della configurazione**: comprende le operazioni per la gestione delle configurazioni del software, come la gestione delle versioni e il controllo delle modifiche;
- **assicurazione della qualità**: questo processo si occupa delle operazioni per assicurare che il software rispetti gli standard di qualità prefissati;
- **verifica**: implica l'analisi e la valutazione dei prodotti software per assicurare che rispondano ai requisiti definiti;
- **validazione**: questo processo si focalizza sulla verifica che il software risponda alle necessità dell'utente e si integri adeguatamente nell'ambiente di lavoro;
- **revisioni congiunte con il cliente**: questo processo coinvolge il cliente nelle operazioni di analisi e valutazione del software;
- **verifiche ispettive interne**: coinvolge il team di sviluppo nelle attività di revisione e valutazione del software;
- **risoluzione dei problemi**: coinvolge l'identificazione e la risoluzione dei problemi nel software.

5.2.3 Processi organizzativi

Questi processi supportano l'organizzazione nel suo insieme e si compongono di:

- **gestione**: risoluzione dei problemi dei processi;
- **gestione delle infrastrutture**: disposizione degli strumenti di assistenza ai processi;
- **gestione dei processi**: manutenzione progressiva dei processi;
- **formazione**: supporto, motivazione e integrazione all'auto-apprendimento;
- **amministrazione**: questo processo riguarda l'amministrazione generale dei processi e delle risorse necessarie per il loro funzionamento.

6 Metriche di qualità

6.1 Metriche per la qualità di processo

Metrica	Nome	Descrizione	Formula
9M-EV	Earned Value	Valore del lavoro effettivamente svolto fino al determinato periodo.	$EV = EAC \times \%lavoro\ svolto$
10M-PV	Planned Value	Stima la somma dei costi realizzativi delle attività imminenti, periodo per periodo.	$PV = BAC \times \%lavoro\ svolto$
11M-AC	Actual Cost	Misura i costi effettivamente sostenuti dall'inizio del progetto fino al presente momento.	Dato reperibile e costantemente aggiornato in "Piano di Progetto v1.0.0"
12M-CV	Cost Variance	Misura la differenza percentuale di budget tra quanto previsto nella pianificazione di un periodo e l'effettiva realizzazione.	$CV = EV - AC$

Metrica	Nome	Descrizione	Formula
13M-EAC	Estimated at Completion	Misura il costo realizzativo stimato per terminare il progetto.	$EAC = BAC \div CPI$
14M-ETC	Estimate to Complete	Stima dei costi realizzativi fino alla fine del progetto.	$ETC = EAC - AC$
24M-CC	Code Coverage	Rappresenta il grado in cui il codice sorgente di un programma è testato.	
28M-PTCP	Passed Test Cases Percentage	Percentuale di casi di test superati.	$PTCP = \frac{Casi\ di\ test\ superati}{Casi\ di\ test\ totali} \times 100$
29M-NCR	Rischi non calcolati	Indica il numero di rischi non calcolati nel documento di "Analisi dei Requisiti v1.0.0".	

Metrica	Nome	Descrizione	Formula
31M-RSI	Requirements Stability Index	Misura impiegata nella quantificazione dell'entità e dell'impatti dei cambiamenti ai requisiti di un progetto.	
32M-SV	Schedule Variance	Indica in percentuale quanto si è in anticipo o in ritardo rispetto alla pianificazione.	$SV = EV - PV$

Tabella 7: Metriche per la qualità di processo

6.2 Metriche per la qualità di prodotto

Metrica	Nome	Descrizione	Formula / Caratteristiche
1M-CRO	Copertura dei Requisiti Obbligatori	Valuta quanto del lavoro svolto durante lo sviluppo corrisponda ai requisiti essenziali o obbligatori definiti in fase di analisi dei requisiti.	

Metrica	Nome	Descrizione	Formula / Caratteristiche
2M-CRD	Copertura dei Requisiti Desidera-bili	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.	
3M-CROP	Copertura dei Requisiti Opzionali	Valuta quanti dei requisiti aggiuntivi, non essenziali o di bassa priorità, sono stati implementati o soddisfatti nel prodotto.	
4M-FU	Facilità di Utilizzo	Metrica che misura l'usabilità di un sistema software.	

Metrica	Nome	Descrizione	Formula / Caratteristiche
5M-TA	Tempo di Apprendimento	Misura il tempo massimo richiesto per apprendere l'utilizzo del prodotto.	Usabilità
22M-IG	Indice Gulpease	Misura la leggibilità di un testo in base alla lunghezza delle parole e delle frasi.	$IG = 89 + \frac{300 \times Numero\ frasi - 10 \times Numero\ lettere}{Numero\ parole}$
23M-CO	Correttezza Ortografica	Misura la presenza di errori ortografici nei documenti.	Affidabilità
24M-CC	Code Coverage	Misura la percentuale di codice sorgente coperto dai test.	
25M-BC	Branch Coverage	Misura la percentuale di rami decisionali coperti dai test.	$BC = \frac{Flussi\ funzionali\ testati}{Flussi\ condizionali\ riusciti\ e\ non} \times 100$
26M-SC	Statement Coverage	Misura la percentuale di statement del codice coperti dai test.	$SC = \frac{Statement\ testati}{Statement\ totali} \times 100$

Metrica	Nome	Descrizione	Formula / Caratteristiche
29M-QMS	Quality Metrics Satisfied	Misura che valuta quante metriche, tra quelle definite, sono state implementate e soddisfatte.	$QMS = \frac{Metriche\ soddisfatte}{Metriche\ totali} \times 100$

Tabella 8: Metriche per la qualità di Prodotto