Norme di Progetto

v2.0

7Last

Versioni

Ver.	Data	Redattore	Verificatore$_G$	Descrizione
2.0	2024-07-16	Elena Ferro	Antonio Benetazzo	Approvazione finale documento
1.4	2024-07-01	Valerio Occhinegro	Raul Seganfreddo	Modificata sezione Piano di Progetto
1.3	2024-06-28	Valerio Occhinegro	Raul Seganfreddo	Completata sezione test
1.2	2024-06-25	Antonio Benetazzo	Valerio Occhinegro	Continuazione sezione test
1.1	2024-06-21	Antonio Benetazzo	Valerio Occhinegro	Aggiunta sezione test
1.0	2024-05-24	Elena Ferro	Valerio Occhinegro	Approvazione finale documento
0.9	2024-05-10	Valerio Occhinegro	Leonardo Baldo	Aggiunta metriche di qualità
0.8	2024-05-08	Matteo Tiozzo	Leonardo Baldo	Aggiunta standard per la qualità
0.7	2024-04-26	Leonardo Baldo	Antonio Benetazzo	Aggiunta processi organizzativi
0.6	2024-04-22	Leonardo Baldo	Antonio Benetazzo	Conclusione processi di supporto
0.5	2024-04-16	Antonio Benetazzo	Valerio Occhinegro	Continuazione processi di supporto
0.4	2024-04-10	Antonio Benetazzo	Valerio Occhinegro	Inizio processi di supporto
0.3	2024-04-08	Antonio Benetazzo	Valerio Occhinegro	Conclusione processi primari
0.2	2024-04-04	Antonio Benetazzo	Valerio Occhinegro	Inizio processi primari
0.1	2024-03-29	Raul Seganfreddo	Valerio Occhinegro	Creazione template e aggiunta introduzione

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 inizierà nelle prime fasi dello svolgimento del progetto, ma proseguirà anche in quelle successive 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 smart city$_G$, 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 stream 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*$_G$ contenente la definizione dei termini tecnici e degli acronimi utilizzati. I termini di questo documento presenti nel glossario$_G$ 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 Normativi

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

1.4.2 Informativi

- **Glossario$_G$** v2.0
  `https://7last.github.io/docs/pb/documentazione-interna/glossario`
- **Documentazione Git**
  `https://git-scm.com/docs` [Ultima consultazione: 2024-05-10]
- **Documentazione Latex**
  `https://www.latex-project.org/help/documentation/` [Ultima consultazione: 2024-05-10]
- **Documentazione Python$_G$**
  `https://docs.python.org/3/` [Ultima consultazione: 2024-05-20]
- **Documentazione Kafka**
  `https://kafka.apache.org/documentation/` [Ultima consultazione: 2024-05-20]
- **Documentazione ClickHouse$_G$**
  `https://clickhouse.com/docs` [Ultima consultazione: 2024-05-22]
- **Documentazione Grafana$_G$**
  `https://grafana.com/docs/grafana/latest/` [Ultima consultazione: 2024-05-16]

- **Documentazione Docker$_G$**
  `https://docs.docker.com/` [Ultima consultazione: 2024-05-13]

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 fasi di seguito specificate.

- **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 milestone$_G$;
  - stesura del piano di progetto$_G$;
  - 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, appunto, 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 Contatti con l'azienda proponente

Sync Lab 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$_G$ sarà redatto un verbale che riassumerà i temi trattati. Tali documenti saranno disponibili al seguente link.

2.1.4 Documentazione fornita

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

2.1.4.1 Piano di qualifica

Il *Piano di Qualifica*$_G$ è un documento che specifica le responsabilità e le attività del verificatore$_G$ nel progetto, delineando le strategie e gli approcci adottati per garantire la qualità del prodotto software in fase di sviluppo. Redatto dal verificatore$_G$, 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 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 affinché il prodotto finale sia di alta qualità;
- **metodologie di testing**: contiene la descrizione di tutti i test necessari a verificare che il prodotto soddisfi i requisiti specificati;
- **cruscotto$_G$** di valutazione della qualità: riporta le attività di verifica svolte e le problematiche riscontrate durante lo sviluppo del software, con l'obiettivo di identificare aree di miglioramento;
- **iniziative di automiglioramento per la qualità**: contiene le azioni correttive e preventive adottate per migliorare la qualità del prodotto e dei processi.

2.1.4.2 Analisi dei requisiti

Questo documento, redatto dall'analista$_G$, offre una panoramica chiara delle richieste e delle aspettative dell'azienda proponente$_G$. Serve come guida per il team di sviluppo, fornendo un elenco dettagliato 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 dalle sezioni qui specificate.

- **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.

- **Elenco dei casi d'uso**: riporta 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**: specifica tutti i vincoli richiesti dalla proponente$_G$ 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$_G$ e soddisfi le necessità degli utenti finali.

2.1.4.3 Piano di progetto

Redatto dal responsabile$_G$, 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 visione chiara e ben definita per il team di progetto. Il documento è costituito principalmente dalle sezioni qui elencate.

- **Analisi dei rischi**: identificazione delle problematiche che potrebbero insorgere durante il processo e che potrebbero arrecare ritardi o impedimenti nella progressione 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.
- **Calendario di progetto**: definisce le tempistiche pianificate per le varie revisioni previste durante lo svolgimento del progetto.
- **Stima costi realizzazione**: espone una previsione dettagliata dei costi di realizzazione del progetto.
- **Gestione del modello e sprint$_G$**: definisce i periodi temporali con eventi e attività correlate, all'interno di un calendario. Per ciascun periodo vengono analizzati i rischi, i costi e le attività sia in fase di pianificazione che in fase di retrospettiva.

- **Retrospettiva generale**: analisi generale di tutto il percorso svolto, ottenuta dall'esperienza accumulata, con particolare attenzione agli aspetti positivi e negativi riscontrati.

2.1.4.4 Glossario

Il *Glossario*$_G$ 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$_G$.

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$_G$ e la proponente$_G$, ma stabilisce anche i termini concordati per la consegna del prodotto finito.

2.1.5 Metriche

Codice	Nome esteso
1M-PV	Planned Value
2M-EV	Earned Value
3M-AC	Actual Cost
4M-SV	Schedule Variance
5M-CV	Cost Variance
6M-CPI	Cost Performance Index
7M-SPI	Schedule Performance Index
8M-EAC	Estimate at Completion
9M-ETC	Estimate to Complete
10M-OTDR	On Time Delivery Rate

Tabella 1: Metriche inerenti il processo di fornitura

2.1.6 Strumenti

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

- **Discord** come piattaforma per le riunioni interne e metodo informale per contattare l'azienda proponente$_G$ tramite messaggistica;
- **Google Meet** utilizzato per le riunioni con la proponente$_G$;
- **Google Slides** come strumento per la creazione di presentazioni;
- **LaTeX** come sistema di preparazione di documenti utilizzato principalmente per la creazione di documenti tecnici e scientifici;
- **ClickUp$_G$** come strumento di gestione del progetto per la pianificazione e il monitoraggio delle attività.

2.2 Sviluppo

2.2.1 Introduzione

Lo standard *ISO/IEC 12207:1995* fornisce un quadro completo e strutturato per la gestione del ciclo di vita del software. 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

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 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$_G$, si cerca di comprendere appieno le esigenze degli stakeholder$_G$, inclusi utenti finali e clienti, al fine di garantire che il prodotto software sviluppato soddisfi le loro aspettative e necessità. L'analisi dei requisiti$_G$ include

tipicamente la raccolta e la documentazione dei requisiti funzionali di vincolo e qualitativi, la definizione dei casi d'uso nonché la prioritizzazione e la tracciabilità dei requisiti lungo l'intero ciclo di vita del software.

2.2.2.2 Scopo

Questo processo pone i seguenti obiettivi principali:

- promuovere una comprensione condivisa tra tutte le parti interessate;
- consentire una stima accurata delle tempistiche e dei costi del progetto;
- fornire ai progettisti$_G$ requisiti chiari e facilmente comprensibili;
- definire gli obiettivi del prodotto al fine di soddisfare le aspettative;
- agevolare l'attività di verifica e di test fornendo indicazioni pratiche di riferimento.

2.2.2.3 Documentazione

Gli analisti$_G$ hanno il compito di redigere l'*Analisi dei Requisiti*$_G$, comprendendo le seguenti sezioni:

- **introduzione**, contente la presentazione e lo scopo del documento stesso;
- **descrizione del prodotto**, in cui è definita un'analisi approfondita del prodotto, includendo:
  - obiettivi del prodotto;
  - architettura del prodotto;
  - funzionalità del prodotto;
  - caratteristiche degli utenti;
- **casi d'uso** con le descrizioni delle funzionalità offerte dal sistema dal punto di vista dell'utente, includendo:
  - attori coinvolti;
  - elenco dei casi d'uso
- **requisiti**, suddivisi in:
  - funzionali;
  - qualitativi;
  - di vincolo;
  - prestazionali.

2.2.2.4 Casi d'uso

I casi d'uso forniscono una descrizione dettagliata delle funzionalità del sistema dal punto di vista degli utenti, delineando come il sistema risponda a specifiche azioni o scenari. Ogni caso d'uso deve includere:

- **identificativo**:

  UC-[identificativo_caso_principale].[identificativo_sotto_caso]

  - **identificativo caso principale**: identificativo numerico del caso d'uso principale;
  - **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;
- **attore principale**: entità esterna che interagisce attivamente con il sistema per soddisfare una propria necessità;
- **precondizioni**: lo stato in cui deve trovarsi il sistema affinché la funzionalità sia disponibile per l'attore;
- **postcondizioni**: lo stato in cui si trova il sistema dopo l'esecuzione dello scenario principale;
- **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);
- **user story**$_G$: una breve descrizione di una funzionalità del software, scritta dal punto di vista dell'utente, che fornisce contesto, obiettivi e valore:
  - L'user story$_G$ 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 attori e il sistema, delineando i vari scenari in cui esso verrà utilizzato. Ogni caso d'uso descrive una sequenza specifica di azioni che gli attori compiono per raggiungere un obiettivo particolare all'interno del sistema. I diagrammi dei casi d'uso aiutano a identificare i requisiti funzionali, fornendo una comprensione chiara e condivisa delle aspettative degli utenti. Inoltre, facilitano la comunicazione tra gli sviluppatori e gli stakeholder$_G$, assicurando che tutte le funzionalità necessarie siano considerate e correttamente implementate nell'applicativo. Di seguito vengono elencati i principali componenti di un diagramma dei casi d'uso.

- **Sistema**: delimita i confini del sistema software, indicando quali funzionalità sono incluse e quali sono esterne ad esso.

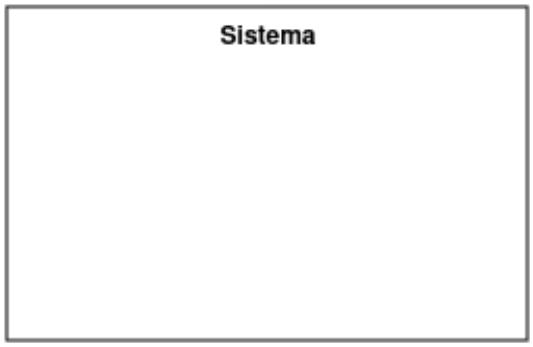

Figura 1: Diagramma dei casi d'uso - sistema

- **Attori**: rappresentano soggetti che interagiscono con il sistema software. Gli attori possono essere persone, altri applicativi o dispositivi che utilizzano le funzionalità del sistema.

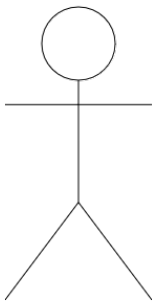

Figura 2: Diagramma dei casi d'uso - attore

- **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.

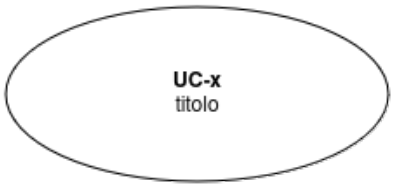

Figura 3: Diagramma dei casi d'uso - caso d'uso

- **Sottocasi d'uso**: rappresentano scenari specifici e dettagliati che si verificano all'interno di un caso d'uso principale, descrivendolo in modo più dettagliato.

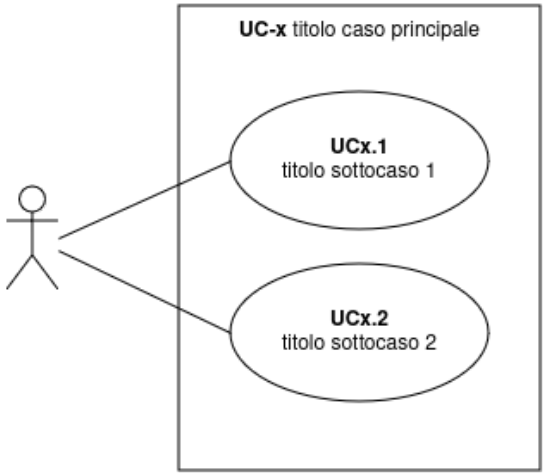

Figura 4: Diagramma dei casi d'uso - sottocaso d'uso

- **Relazioni tra Attori e Casi d'Uso**
  - **Associazione**: è la relazione fondamentale che collega un attore a un caso d'uso, indica quali attori interagiscono con quali casi d'uso.

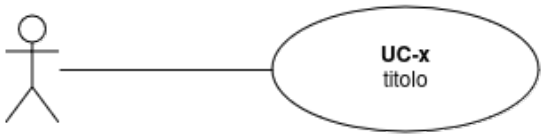

Figura 5: Diagramma dei casi d'uso - associazione

- **Relazioni tra Attori**
  - **Generalizzazione**: rappresenta una relazione in cui un attore figlio condivide almeno le funzionalità di un attore genitore. Utilizzata quando diversi attori condividono comportamenti comuni.

Figura 6: Diagramma dei casi d'uso - generalizzazione tra attori

- **Relazioni tra Casi d'Uso**
  - **Inclusione**: rappresenta una dipendenza in cui il comportamento del caso d'uso incluso è incorporato ogni volta che viene eseguito il caso d'uso base. Il caso d'uso incluso contiene funzionalità che sono riutilizzate in più casi d'uso principali, permettendo di evitare la duplicazione di comportamenti comuni.

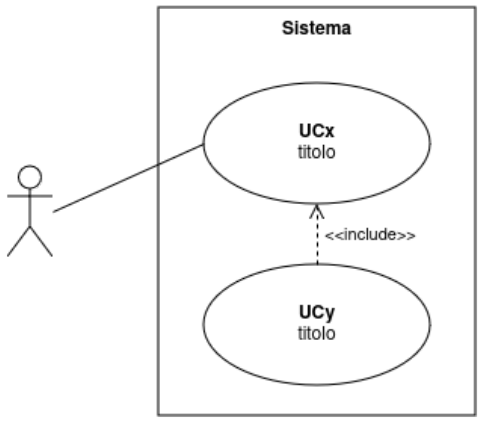

Figura 7: Diagramma dei casi d'uso - inclusione

– **Estensione**: mostra una relazione in cui un caso d'uso esteso aumenta le funzionalità del caso d'uso principale. Il caso d'uso esteso viene eseguito solo sotto determinate condizioni, interrompendo l'esecuzione del caso d'uso principale.

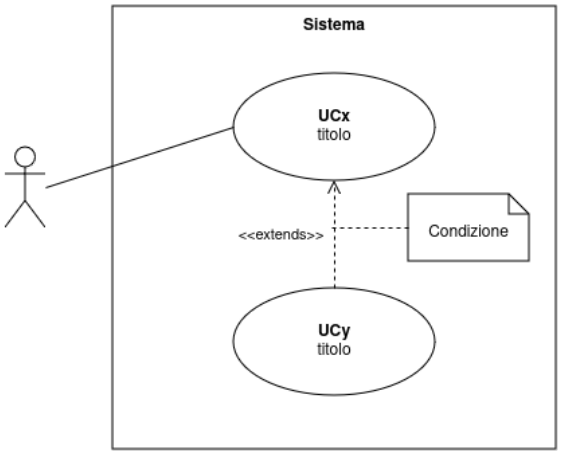

Figura 8: Diagramma dei casi d'uso - estensione

– **Generalizzazione**: rappresenta una relazione in cui un caso d'uso figlio può aggiungere funzionalità o modificare il comportamento di un caso d'uso genitore. Tutte le funzionalità definite nel caso d'uso genitore si mantengono nel caso d'uso figlio se queste non vengono ridefinite.

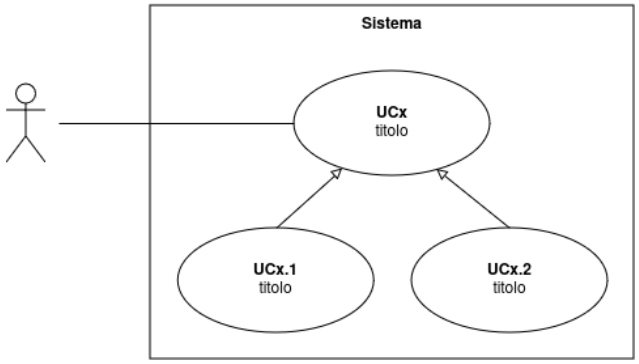

Figura 9: Diagramma dei casi d'uso - generalizzazione tra casi d'uso

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$_G$. 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 tre categorie principali:

- **requisiti funzionali**, i quali descrivono le azioni specifiche che il sistema deve essere in grado di eseguire;
- **requisiti non funzionali**, che definiscono le proprietà e le caratteristiche del sistema che ne influenzano il funzionamento e l'uso;
- **requisiti di vincolo**, che rappresentano i vincoli e le restrizioni che il sistema deve rispettare.

Ogni requisito è costituito da:

1. **codice** - i requisiti sono codificati nel seguente modo:

   R[tipologia]-[identificativo]

   dove:

   - **[tipologia]** indica la tipologia del requisito, che può essere
     - **F** = requisito funzionale
     - **Q** = requisito di qualità

  - **V** = requisito di vincolo;
- **[identificativo]** è un numero progressivo che identifica univocamente il requisito;

2. **importanza** - indica il grado di importanza del requisito, che può essere:
   - **obbligatorio**, ovvero irrinunciabile per il committente$_G$;
   - **desiderabile** cioè 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$_G$** indica i requisiti individuati a seguito dell'analisi dello stesso;
   - **interno** identifica i requisiti individuati durante le riunioni interne e da coloro che hanno il ruolo di analista$_G$;
   - **esterno** segnala i requisiti aggiuntivi individuati in seguito a incontri con la proponente$_G$;
   - **piano di qualifica$_G$** evidenzia i requisiti necessari per adeguare il prodotto agli standard di qualità definiti nel documento *Piano di Qualifica$_G$*;
   - **norme di progetto$_G$** specifica i requisiti necessari per adeguare il prodotto alle norme stabilite nel documento *Norme di Progetto$_G$*.

2.2.2.7 Metriche

Nell'analisi dei requisiti$_G$, le metriche sono strumenti utilizzati per misurare vari aspetti del processo di sviluppo e del prodotto finale. All'interno di un'analisi dei requisiti$_G$, 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$_G$.

Codice	Nome esteso
11M-PRO	Percentuale requisiti obbligatori
12M-PRD	Percentuale requisiti desiderabili
13M-PRO	Percentuale requisiti opzionali

Tabella 2: Metriche inerenti i requisiti

2.2.2.8 Strumenti

Draw.io è uno strumento online che permette di creare diagrammi dei casi d'uso in modo semplice e intuitivo. Questo strumento offre una vasta gamma di funzionalità e opzioni di personalizzazione per creare diagrammi chiari e ben strutturati. Draw.io è un'applicazione web-based, quindi non richiede alcun download o installazione, rendendo facile e veloce la creazione di diagrammi dei casi d'uso.

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$_G$ 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 soddisfare i requisiti attraverso un sistema di qualità definito dall'architettura del prodotto.

- **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.

Lo sviluppo del progetto inizia con l'analisi delle tecnologie suggerite dalla proponente$_G$ e un confronto con eventuali alternative, valutando i pro e i contro di ciascuna. Dopo aver selezionato le tecnologie più adatte, si procede con lo sviluppo di un Proof of Concept$_G$ (PoC$_G$), un prototipo grezzo che dimostra l'idoneità delle tecnologie scelte e la capacità del team di utilizzarle efficacemente. Questo PoC$_G$ costituisce la componente principale della Technology Baseline, che funge da base per lo sviluppo del prodotto finale. A partire da questa, si procede con lo sviluppo del prodotto vero e proprio, trasformando il PoC$_G$ in un Minimum Viable Product$_G$ (MVP$_G$), ovvero un prodotto minimale che soddisfa i requisiti minimi richiesti e che diventerà parte integrante della Product Baseline$_G$ (PB$_G$).

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;

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.
- **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 performante 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

- **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$_G$.

- **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$_G$ 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$_G$.

2.2.3.5.1 Diagrammi delle Classi

Un tipo di diagramma UML ampiamente utilizzato nel processo di progettazione del software è il diagramma delle classi, che rappresenta la struttura statica del sistema, mostrando le classi, gli attributi, i metodi e le relazioni tra di esse. 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] = default

dove:

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

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

visibilità nome(parametri): ritorno

dove:

- **visibilità** indica il livello di accesso ai metodi, che può essere
  - **+** = pubblico
  - **-** = privato
  - **#** = protetto
  - **~** = package;
- **nome**, indica il nome del metodo, deve essere rappresentativo, chiaro e deve seguire la notazione *nomeMetodo*(*parametri*)*: tipoRitorno*;
- **parametri** rappresenta l'elenco dei parametri del metodo, separati tramite virgola, dove ogni parametro deve seguire la notazione *nomeParametro: tipo*;
- **ritorno** indica il tipo di dato restituito dal metodo.

Convenzioni sui metodi

Le indicazioni dei metodi all'interno del diagramma delle classi seguono le seguenti convenzioni:

- **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.

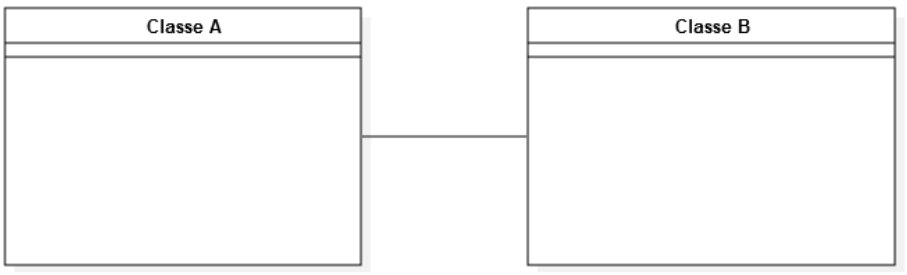

Figura 10: Diagramma delle classi - associazione

- **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.

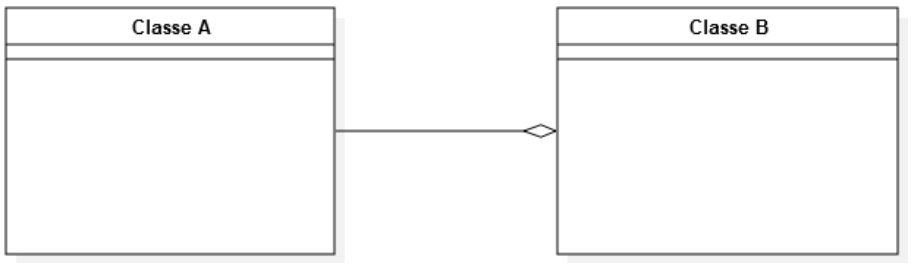

Figura 11: Diagramma delle classi - aggregazione

- **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.

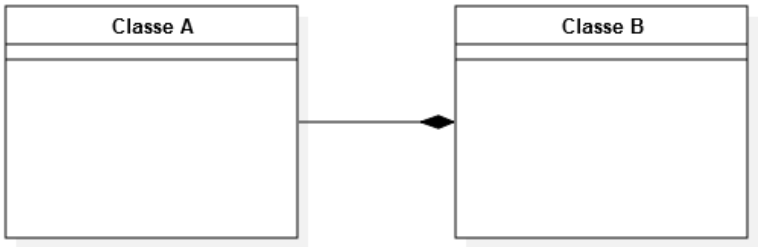

Figura 12: Diagramma delle classi - composizione

- **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.

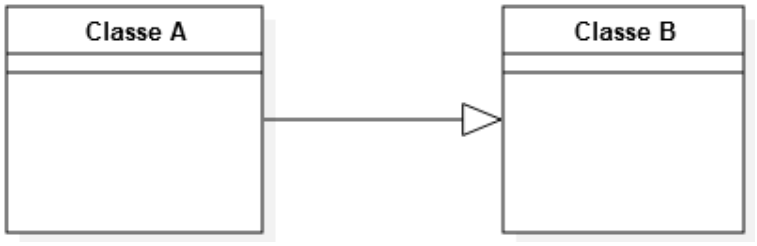

Figura 13: Diagramma delle classi - generalizzazione

- **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.

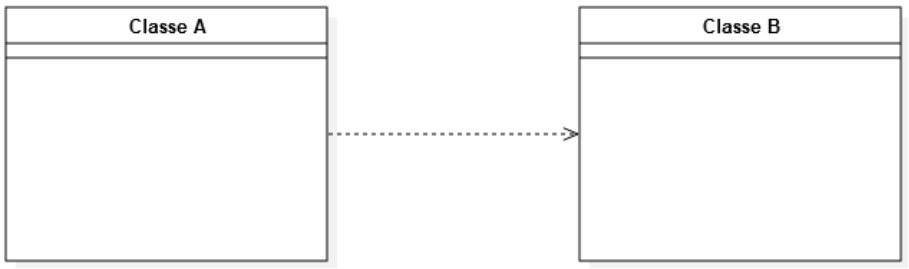

Figura 14: Diagramma delle classi - dipendenza

- **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.

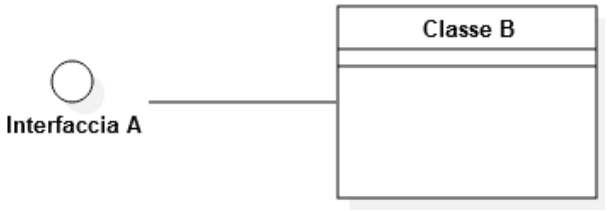

Figura 15: Diagramma delle classi - realizzazione

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.

2.2.3.8 Metriche

Codice	Nome esteso
14M-PG	Profondità delle Gerarchie

Tabella 3: Metriche inerenti la progettazione

2.2.3.9 Strumenti

Draw.io: è uno strumento per la creazione di diagrammi online gratuito che consente di creare, modificare e condividere diagrammi di flusso, mappe mentali, organigrammi e altri tipi di diagrammi grafici.

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 l' analisi e la 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$_G$.

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 implementato nel modo più leggibile ed autoesplicativo possibile. I commenti verranno impiegati solo ove strettamente necessari;
- **indentazione e formattazione consistente**: il codice deve essere correttamente indentato e formattato, mediante l'uso di tabulazioni, per garantire una corretta leggibilità e comprensione ;

- **gestione delle eccezioni**: il codice deve gestire correttamente le eccezioni e gli errori, fornendo messaggi significativi e gestendo le situazioni anomale in modo appropriato;
- **riuso del codice**: il codice deve essere scritto in modo modulare e riutilizzabile;
- **testabilità**: scrivere funzioni piccole e focalizzate, ad esempio, aiuta a garantire che il codice sia facilmente testabile;
- **sicurezza**: il codice deve essere scritto in modo sicuro, evitando vulnerabilità e falle di sicurezza;
- **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 Metriche

Codice	Nome esteso
15M-PPM	Parametri per Metodo
16M-CPC	Campi per classe
17M-LCPM	Linee di codice per metodo
18M-CCM	Complessità Ciclomatica

Tabella 4: Metriche riguardanti l'attività di codifica

2.2.4.5 Strumenti

Visual Studio Code: è un ambiente di sviluppo integrato (IDE) sviluppato da Microsoft e adottato dal gruppo per lo sviluppo del software. Offre funzionalità avanzate per la scrittura, la modifica e il debug del codice, con supporto per numerosi linguaggi di programmazione e framework.

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

L'utilizzo del file Docker$_G$ 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$_G$.

- **Chiarezza e semplicità**: sono principi fondamentali nella scrittura di un Dockerfile. Questi principi aiutano a creare immagini Docker$_G$ 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$_G$. 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 prestazioni, i file Docker$_G$ 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$_G$ 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$_G$. La gestione efficiente di questi strati è cruciale poiché influisce direttamente sulle dimensioni dell'immagine Docker$_G$ finale e sulla velocità di build.
- **Riduzione delle dimensioni delle immagini**: la riduzione delle dimensioni delle immagini Docker$_G$ 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$_G$ risultante sia configurata correttamente e funzioni come previsto.

2.2.5.2 Strumenti

- **Docker$_G$**: 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. 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, poiché 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;
- garantire uno standard di qualità all'interno dei processi produttivi.

Lo scopo della sezione è:

- definire un insieme di regole e convenzioni per garantire la coerenza e la qualità della documentazione prodotta;
- creare template per ogni tipologia di documento così da garantire omogeneità.

3.1.2 Versionamento

La documentazione viene versionata allo stesso modo del codice sorgente, utilizzando la piattaforma Github$_G$. Ciò permette di tenere traccia delle modifiche apportate ai documenti e di realizzare procedure automatiche per l'individuazione degli errori grammaticali, inserimento della "G" a pedice per i termini presenti nel glossario$_G$ e la pubblicazione nel sito del progetto.
Abbiamo stabilito di utilizzare il branch `main` per la pubblicazione dei documenti approvati dal verificatore$_G$ e il branch `develop` per contenere i file in formato LaTex da cui derivano i documenti finali.

3.1.2.1 Branch policy

Sono state definite le seguenti regole per la gestione dei branch:

- `main`: viene aggiornato solamente dalla *Github*$_G$ Action che si occupa di pubblicare i documenti approvati presenti nel branch `develop`;

- `develop`: per la pubblicazione in questo branch è necessario ricevere un'approvazione da parte del verificatore$_G$ tramite *pull request*.

3.1.3 Tipografia e sorgente documenti

Per la redazione dei documenti abbiamo deciso di utilizzare il linguaggio di markup LaTeX, grazie alla sua flessibilità e possibilità di creare documenti di alta qualità tipografica. Inoltre, abbiamo stabilito di suddividere ciascun documento in file differenti, in modo da consentire ai membri del team di lavorare contemporaneamente su sezioni diverse, riducendo il numero di conflitti e semplificando la gestione del versionamento.

3.1.4 Ciclo di vita

Ciascun documento segue un ciclo di vita ben definito, che prevede le seguenti fasi:

1. **Creazione branch** a partire da `develop`, utilizzando Gitflow [Ultima consultazione 2024-05-13];
2. **Scelta del template** da utilizzare per il documento, scegliendo tra *documento*, *verbale interno* o *verbale esterno*;
3. **Redazione** del documento, seguendo le norme tipografiche e le convenzioni stabilite;
4. **Commit** delle modifiche effettuate;
5. **Apertura *pull request*** per la revisione del documento, che dovrà essere approvata dal verificatore$_G$ e dal responsabile$_G$ per lo sprint$_G$ in corso;
6. **Chiusura *pull request*** e **merge** del branch;
7. **Completamento task** su *ClickUp*$_G$.

3.1.5 Convenzioni nomenclatura e struttura di archiviazione

Tutti i documenti prodotti devono seguire la seguente convenzione di nomenclatura ben definita, in modo da garantire una corretta organizzazione e archiviazione dei file

`nome_fase/tipo_documento/nome_documento/nome_documento.tex`

Dove:

- **nome_fase**: rappresenta la fase di progetto in cui il documento è stato prodotto, può assumere i seguenti valori:
  - **1_candidatura**;
  - **2_RTB**;
  - **3_PB**.
- **tipo_documento**: rappresenta la tipologia di documento, può assumere i seguenti valori:
  - **documentazione_interna**;
  - **documentazione_esterna**;
  - **verbali_interni**;
  - **verbali_esterni**.
- **nome_documento**: rappresenta il nome del documento, scritto in snake case.

3.1.6 Procedure correlate alla redazione di documenti

3.1.6.1 I redattori

Il redattore si occupa della stesura del documento, seguendo le norme tipografiche e le convenzioni stabilite, in modo da garantire la chiarezza e la coerenza del testo. Dovrà rispettare le convenzioni sopracitate.

3.1.6.2 Il resposabile

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

- definire quali documenti devono essere prodotti e quali compiti devono essere svolti;
- gestisce le risorse e gli strumenti necessari per ultimare tali attività;
- approva la versione finale dei documenti;
- comunica con gli stakeholder$_G$.

3.1.6.3 L'amministratore

L'amministratore$_G$ è colui che si occupa della configurazione degli strumenti di supporto alla produzione del software; nel nostro caso, inserisce le attività nell'ITS ClickUp$_G$ e le assegna ai membri del gruppo.

3.1.6.4 I verificatori

Il compito del verificatore$_G$ è di controllare la correttezza e la qualità dei documenti prodotti, assicurandosi che siano conformi agli standard e alle norme stabilite.

3.1.7 Struttura del documento

Tutti i documenti produtti devono seguire una struttura ben definita, che prevede le componenti in seguito elencate.

3.1.7.1 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.

3.1.7.2 Registro delle modifiche

La seconda pagina è dedicata al registro delle modifiche, rappresentato mediante una tabella. Il suo scopo è quello di tenere traccia di tutti i cambiamenti effettuati al documento, in modo da garantire la rintracciabilità e la trasparenza delle modifiche. Ogni riga della tabella corrisponde a un cambiamento, e contiene le seguenti informazioni:

- **versione** del documento, nel formato vX.Y, dove X rappresenta il numero di versione principale e Y il numero di versione secondaria;
- **data** di ultima modifica;

- nome del **redattore**;
- nome del **verificatore**$_G$;
- **descrizione** della modifica.

3.1.7.3 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 nei documenti sono visibili negli appositi indici e sono direttamente accessibili tramite link ipertestuali.

3.1.7.4 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.

3.1.7.5 Struttura dei verbali

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 per entrambe le tipologie, ad eccezione dell'ultima pagina.

- **prima pagina**:
  - tipologia verbale (interno / esterno);
  - titolo del documento;
  - sottotitolo del documento, "Riunione interna settimanale" per i verbali interni e "Colloquio con *Sync Lab*" per i verbali esterni;
  - data e versione del documento;
  - logo del gruppo
  - nome del gruppo.
- **Registro delle modifiche**;

- **Dettagli sulla riunione**:
  - sede della riunione (piattaforma utilizzata);
  - orario di inizio e fine;
  - partecipanti del gruppo (interni) con specificazione del ruolo;
  - 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;
  - **verbale**: resoconto dettagliato dell'incontro, con una sottosezione per ciascun punto dell'ordine del giorno, gli obiettivi prefissati e le decisioni prese;
- **Ultima pagina**: solo in caso di verbale esterno, l'ultima pagina contiene data e firma di un componente esterno, presente durante l'incontro.

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

3.1.8 Norme tipografiche

Nomi dei file
I nomi dei documenti devono essere coerenti con il loro contenuto, scritti in snake case. Nel branch `develop` tutti i documenti non contengono la versione nel nome, al fine di evitare conflitti durante la modifica. Nel momento in cui il documento viene approvato, esso viene spostato nel branch `main` e la versione viene aggiunta automaticamente al nome del file.
In particolare, la denominazione dei file nel branch `main` deve seguire la seguente convenzione:

- **Verbali**: verbale_esterno/interno_YYYY_MM_DD_vA.B;
- **Norme di Progetto**$_G$: norme_di_progetto_vA.B;
- **Analisi dei Requisiti**$_G$: analisi_dei_requisiti_vA.B;
- **Piano di Progetto**$_G$: piano_di_progetto_vA.B;
- **Lettera di Presentazione**: lettera_di_presentazione;
- **Glossario**$_G$: glossario$_G$_vA.B.

Stile del testo

- **Grassetto**:
  - titoli di sezione;
  - termini importanti;
  - parole seguite da descrizione o elenchi puntati.
- **Corsivo**:
  - nome del gruppo e dell'azienda proponente$_G$;
  - riferimenti a documenti esterni.
- **Maiuscolo**:
  - acronimi;
  - iniziali dei nomi;

Regole sintattiche:

- negli elenchi ogni voce deve terminare con ";" mentre l'ultima ".";
- 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;

3.1.9 Metriche

Codice	Nome esteso
19M-IG	Indice di Gulpease
20M-CO	Correttezza Ortografica

Tabella 5: Metriche inerenti il processo di documentazione

3.1.10 Strumenti

Gli strumenti utilizzati per la redazione dei documenti sono:

- **LaTeX**: impiegato nella stesura dei documenti;
- **GitHub**$_{\textcircled{G}}$: utilizzato per la gestione delle versioni e per la condivisione dei documenti;
- **Visual Studio Code**: utilizzato come editor di testo per la scrittura dei documenti.

3.2 Accertamento della qualità

3.2.1 Introduzione

L'accertamento della qualità consiste in un insieme di attività e processi volti a garantire che il software sviluppato soddisfi gli standard di qualitativi richiesti. Questo include l'adesione a requisiti concordati con la proponente$_G$, normative ed è cruciale per assicurare che il prodotto finale sia affidabile e performante.

3.2.2 Attività

Il processo di accertamento della qualità comprende le seguenti attività:

1. **sviluppo del piano di qualifica$_G$**: vengono definiti gli standard di qualità e le metriche che il progetto deve raggiungere;
2. **monitoraggio continuo 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. **gestione delle configurazioni**: attraverso un sistema di gestione delle configurazioni ci si assicura che ciascuna di esse venga sottoposta ad opportuni controlli e valutazioni;
4. **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;
5. **miglioramento continuo**: vengono applicate le modifiche ai processi basate sui feedback e sulle analisi per migliorare continuamente la qualità del software e dei processi di sviluppo;
6. **confronto con la proponente$_G$**: viene instaurata una comunicazione con la proponente$_G$ per ricevere feedback costante sul lavoro svolto.

3.2.3 Piano di qualifica

Il documento *Piano di Qualifica*$_G$ è redatto per garantire che il software sviluppato rispetti gli standard di qualità richiesti e soddisfi le aspettative degli stakeholder$_G$. 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.

3.2.4 Ciclo di Deming

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

- **plan**: in cui si definiscono le attività, scadenze, responsabilità e risorse per raggiungere gli 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.

È importante specificare che questo ciclo **non opera** sul prodotto, bensì sul ***way of working*** del gruppo, per migliorarlo e renderlo più efficiente.

3.2.5 Struttura e identificazioni metriche

Ogni metrica presenta la seguente struttura:

- **metrica** corrisponde al codice identificativo nel formato:

  [numero]M-[acronimo]

  dove:

  - **[numero]** = numero progressivo univoco per ogni metrica
  - **M** = metrica
  - **[acronimo]** = abbreviazione del nome della metrica;
- **nome** rappresenta il nome della metrica;
- **definizione** descrive il significato e lo scopo della metrica;
- **come si calcola** fornisce le istruzioni per il calcolo della metrica;
- **valore ammissibile** identifica il valore minimo affinché la metrica sia considerabile soddisfacente e conforme agli obiettivi di qualità;
- **valore ottimo** corrisponde al valore ottimale e ideale che dovrebbe essere raggiunto dalla metrica.

3.2.6 Metriche

Codice	Nome esteso
21M-FU	Facilità di utilizzo
22M-TA	Tempo di apprendimento
23M-TR	Tempo di risposta
24M-TE	Tempo di elaborazione
25M-QMS	Metriche di qualità soddisfatte

Tabella 6: Metriche relative all'accertamento della qualità

3.3 Verifica

3.3.1 Introduzione

La verifica nel ciclo di vita del software garantisce l'efficienza e la correttezza delle attività, assicurando che i prodotti soddisfino i requisiti. I verificatori$_G$ applicano tecniche di test seguendo procedure definite all'interno del Piano di Qualifica$_G$, il quale traccia il percorso di verifica, stabilendo obiettivi e criteri di accettazione. Tutti i processi attivi vengono sottoposti a verifica dopo aver raggiunto un buon grado di completamento o a seguito di modifiche dello stato del processo stesso.

3.3.2 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$_G$ 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 con l'ausilio di controlli automatici;
- **coerenza** - ci si assicura che i termini e le definizioni siano utilizzati in modo uniforme e coerente in tutto il documento.

3.3.2.1 Processo di revisione

Al momento di completamento delle modifiche in un documento il redattore è tenuto a creare una pull request dal branch in cui ha effettuato le modifiche verso il branch `develop`, richiedendo una revisione da parte del verificatore$_G$ e del responsabile$_G$; in caso di approvazione verrà effettuato il merge e la issue creata su ClickUp$_G$ verrà spostata da *in progres* a *done*.

3.3.3 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**.

3.3.3.1 Analisi dinamica

L'analisi dinamica riguarda l'esame del comportamento del software durante la sua esecuzione. Questo tipo di analisi viene eseguito a tempo di esecuzione 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; viene effettuata tramite l'utilizzo delle seguenti tipologie di test:

- test di unità;
- test di integrazione;
- test di sistema;
- test di regressione;
- test di accettazione.

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 i test elencati in precedenza.

3.3.4 Testing

Il testing è una fase fondamentale 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;
- **stato del test** - indicazione dello stato del test, che può essere "superato", "non superato" o "non implementato".

3.3.4.1 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, permettendo 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.

3.3.4.2 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.

3.3.4.3 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.

3.3.4.4 Test di regressione

I test di regressione assicurano che le modifiche apportate al codice non abbiano introdotto nuovi difetti o intaccato 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 il corretto funzionamento anche dopo tali modifiche.

3.3.4.5 Test di accettazione

I test di accettazione sono una fase nel processo di sviluppo del software durante la quale il prodotto viene testato per valutare se soddisfa i requisiti e le aspettative degli

utenti finali, dei clienti o degli stakeholder$_G$. Questi test vengono eseguiti dopo che il software è stato completamente sviluppato e prima del suo rilascio ufficiale, consentendo agli utenti finali di valutarlo in un ambiente simile a quello di produzione e fornire feedback sulle sue funzionalità, l'usabilità e la conformità ai requisiti.

3.3.4.6 Sequenza delle fasi di test

I test vengono eseguiti in sequenza, partendo da quelli di unità e procedendo verso quelli 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.

3.3.4.7 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 l'appartenenza del test, può assumere i valori
  - **U** = test di unità
  - **I** = test di integrazione
  - **S** = test di sistema
  - **R** = test di regressione
  - **A** = test di accettazione.

3.3.4.8 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$_G$"*, in particolare nella sezione *"Metodologie di Testing"*.

3.3.4.9 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.

Nell'ambito dell'analisi statica del software, l'inspection e il walkthrough sono tecniche di revisione del codice e della documentazione, ma presentano differenze significative in termini di obiettivi, formalità e procedure.

3.3.4.9.1 Inspection

L'inspection è una tecnica formale e strutturata di revisione del codice. Le sue principali caratteristiche sono:

- **formalità** - l'inspection segue un processo rigoroso con ruoli definiti (moderatore, autore, lettore, ispettore, e registratore) e fasi ben precise (pianificazione, overview, preparazione, meeting di inspection, rework, e follow-up);
- **obiettivi** - l'obiettivo principale è identificare difetti nel codice o nella documentazione. Si concentra su aspetti come errori di logica, violazioni degli standard di codifica, problemi di performance e sicurezza;

- **documentazione** - viene prodotta una documentazione dettagliata delle osservazioni e dei difetti riscontrati, che serve come base per la correzione e per tracciare le azioni successive;
- **ruoli e responsabilità** - ogni partecipante ha un ruolo specifico e le attività sono strettamente coordinate. Il moderatore gestisce il processo, l'autore presenta il lavoro, gli ispettori individuano i difetti, il lettore guida la revisione del materiale, e il registratore annota i difetti trovati;
- **preparazione** - i partecipanti devono esaminare il materiale da rivedere in anticipo e preparare una lista di potenziali problemi.

3.3.4.9.2 Walkthrough

Il walkthrough è una tecnica meno formale e più flessibile rispetto all'inspection. Le sue principali caratteristiche sono:

- **informalità** - il walkthrough è meno strutturato e può essere condotto in modo informale. Non richiede ruoli rigidamente definiti o una procedura rigorosa;
- **obiettivi** - l'obiettivo principale è comprendere il codice o la documentazione, discutere possibili miglioramenti e condividere conoscenze tra i membri del team. Anche se l'individuazione dei difetti è uno degli scopi, non è l'unico focus;
- **documentazione** - la documentazione delle osservazioni è meno formale e dettagliata rispetto all'inspection. Può non essere sempre prodotta una documentazione completa delle discussioni e dei difetti riscontrati;
- **ruoli e responsabilità** - non ci sono ruoli formali assegnati. Solitamente, l'autore del codice o della documentazione guida la revisione e i partecipanti contribuiscono con commenti e suggerimenti;
- **preparazione** - la preparazione è meno intensiva. Non è necessario che tutti i partecipanti esaminino il materiale in anticipo, anche se può essere utile.

3.3.5 Metriche

Codice	Nome esteso
26M-CC	Code Coverage
27M-BC	Branch Coverage
28M-SC	Statement Coverage
29M-FD	Failure Density
30M-PTCP	Passed Test Case Percentage

Tabella 7: Metriche inerenti il processo di verifica

3.3.6 Strumenti

- Aspell per il controllo ortografico;
- unittest libreria di Python$_G$ per lo sviluppo di test.

3.4 Validazione

3.4.1 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.

3.4.2 Procedura di validazione

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

- l'implementazione di tutti i casi d'uso;
- la conformità del prodotto ai requisiti obbligatori;
- il soddisfacimento di altri requisiti concordati con il committente$_G$.

3.5 Gestione della configurazione

3.5.1 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. 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.

3.5.2 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_G, PB_G ed eventuale CA_G;
- **Y** rappresenta una versione intermedia e viene incrementata ad ogni modifica significativa del documento.

3.5.3 Repository

Il team utilizza due repository:

- docs, contenente la documentazione prodotta;
- $SyncCity_G$, contenente il codice del progetto.

3.5.3.1 Struttura repository

Il repository inerente alla documentazione è così organizzato:

- **1_candidatura**:
  - **verbali_esterni**: al suo interno sono presenti i verbali delle riunioni avute con i vari proponenti fino all'approvazione della candidatura;
  - **verbali_interni**: contenente i verbali delle riunioni interne svolte fino all'approvazione della candidatura;
  - **lettera_di_presentazione**: documento di presentazione per la candidatura al $capitolato_G$ scelto;

  - **preventivo_costi_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**: contenente i seguenti documenti:
    - **analisi_dei_requisiti_v1.0**;
    - **piano_di_progetto_v1.0**;
    - **piano_di_qualifica_v1.0**;
  - **documentazione_interna**: contenente i seguenti documenti:
    - **glossario_v1.0**;
    - **norme_di_progetto_v1.0**;
  - **verbali_esterni**: al suo interno sono presenti i verbali delle riunioni avute con i membri di *Sync Lab S.r.l.* dal momento della candidatura alla revisione RTB;
  - **verbali_interni**: contenente i verbali delle riunioni interne svolte dal momento della candidatura alla revisione RTB;

- **3_PB**:
  - **documentazione_esterna**: contenente i seguenti documenti:
    - **analisi_dei_requisiti_v2.0**;
    - **piano_di_progetto_v2.0**;
    - **piano_di_qualifica_v2.0**;
    - **specifica_tecnica_v1.0**;
    - **manuale_utente_v1.0**;
  - **documentazione_interna**: contenente i seguenti documenti:
    - **glossario_v2.0**;
    - **norme_di_progetto_v2.0**.
  - **verbali_esterni**: al suo interno sono presenti i verbali delle riunioni avute con i membri di *Sync Lab S.r.l.* dalla revisione RTB alla revisione PB;
  - **verbali_interni**: contenente i verbali delle riunioni interne svolte dalla revisione RTB alla revisione PB.

3.5.4 Sincronizzazione e branching

3.5.4.1 Documentazione

L'approccio adottato per la redazione della documentazione segue il workflow noto come *Gitflow*. 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).

3.5.4.2 Sviluppo

7Last utilizza Gitflow come flusso di lavoro.

Flusso di lavoro Gitflow:

1. `main` **branch**
   - branch principale e stabile;
   - contiene il codice in produzione;
   - ogni commit rappresenta una versione rilasciata;
2. `develop` **branch**
   - branch per lo sviluppo integrato;
   - riceve i merge dai feature branches;
   - rappresenta lo stato intermedio prima del rilascio;
3. **feature branch**
   - derivano dal branch `develop`;

- utilizzati per sviluppare nuove funzionalità;
- dopo il completamento, viene fatto il merge nel branch `develop`;

4. **release branch**
   - derivano dal branch `develop`;
   - preparano una nuova versione per il rilascio;
   - permettono di effettuare correzioni di bug e preparare la documentazione;
   - dopo il completamento, viene fatto il merge sia nel branch `main` che nel branch `develop`;
5. **hotfix branch**
   - derivano dal branch master;
   - utilizzati per correggere rapidamente bug critici in produzione;
   - dopo il completamento, viene fatto il merge sia nel branch `main` che nel branch `develop`.

3.5.5 Consegna e rilascio

Il processo di consegna e rilascio 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_G (Requirements and Technology Baseline$_G$), mentre la creazione delle successive release deve avvenire in concomitanza con la baseline PB_G (Product Baseline$_G$) e, se prevista, CA_G (Customer Acceptance$_G$).

3.5.6 Sito del gruppo

Il gruppo ha sviluppato un sito disponibile al link: `https://7last.github.io/`; che ha lo scopo di facilitare la consultazione dei documenti redatti fino ad ora. Tale pagina web viene aggiornata automaticamente per rispecchiare lo stato dei documenti presenti

all'interno del branch `main`. Inoltre facilita la consultazione del glossario$_G$ grazie a dei link, presenti nelle parole distinte dalla lettera G al pedice, diretti alla definizione del termine.

3.5.7 Strumenti

- **Github**$_G$: utilizzato per la gestione delle versioni e per la condivisione dei documenti;
- **ClickUp**$_G$: è una piattaforma di gestione del lavoro all-in-one progettata per pianificare, organizzare e collaborare su progetti e attività.

3.6 Analisi congiunta

3.6.1 Introduzione

L'analisi congiunta è un'attività collaborativa cruciale in cui i recensori e i recensiti si incontrano per esaminare e valutare diversi aspetti del progetto. Nel nostro caso i recensori sono costituiti da proponente$_G$, committente$_G$ e stakeholder$_G$; mentre i recensiti sono rappresentati dal gruppo *7Last*. 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 fissati.

3.6.2 Realizzazione del processo

Il processo comprende i seguenti impegni:

3.6.2.1 Revisioni periodiche

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

3.6.2.2 Stato avanzamento lavori

Al termine di ogni sprint$_G$ viene svolta una revisione SAL$_G$ (Stato Avanzamento Lavori$_G$) tra il team e la proponente$_G$. Questa revisione ha lo scopo di valutare il lavoro svolto durante lo sprint$_G$ 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$_G$ successivo.

3.6.2.3 Revisioni straordinarie

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

3.6.2.4 Risorse per le revisioni

Le risorse coinvolte nelle revisioni possono essere differenti, ad esempio: strumenti hardware, software, documentazione... è fondamentale che tali risorse siano discusse e concordate tra tutte le parti coinvolte.

3.6.2.5 Organizzazione degli incontri

Pochi giorni prima della riunione con la proponente$_G$ *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;

3.6.2.6 Documenti prodotti e decisioni approvate

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.

3.7 Risoluzione dei problemi

3.7.1 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.

3.7.2 Gestione dei rischi

All'interno del documento *Piano di Progetto*$_G$, 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.

3.7.2.1 Codifica dei rischi

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

R[tipologia]-[indice]

dove:

- **[tipologia]** identifica la categoria di rischio, la quale può essere organizzativa (**O**), tecnologica (**T**) o comunicativa (**C**);
- **[indice]** rappresenta un valore numerico incrementale che identifica univocamente il rischio per ogni tipologia.

3.7.3 Metriche

Metrica	Nome
31M-RMR	Risk Mitigation Rate
32M-NCR	Rischi Non Calcolati

Tabella 8: Metriche relative alla risoluzione dei problemi

3.7.4 Strumenti

- **ClickUp**$_{\ominus}$: utilizzato per la gestione delle attività e delle issue;

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

La gestione dei processi è un'attività chiave per garantire che il progetto sia completato con successo e in conformità con gli obiettivi e i requisiti stabiliti. Questa fase del progetto si concentra sulla pianificazione, organizzazione, monitoraggio e controllo delle attività coinvolte nel ciclo di vita del software, assicurando che il lavoro svolto sia di alta qualità e soddisfi le aspettative del cliente. Le attività di gestione dei processi sono qui di seguito specificate.

- **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.
- **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.
- **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$_G$. 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. La pianificazione dei processi è essenziale per garantire che il software 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. 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, svolgendo le mansioni previste e assumendosi le responsabilità proprie del determinato ruolo, di seguito meglio specificate.

Responsabile$_G$:

- coordina il gruppo di lavoro;
- pianifica e controlla le attività;
- gestisce le risorse;

- gestisce le comunicazioni con l'esterno;
- redige il *Piano di Progetto*$_G$.

Amministratore$_G$:

- gestisce l'ambiente di lavoro;
- gestisce le procedure e le norme;
- gestisce la configurazione del prodotto;
- redige le *Norme di Progetto*$_G$.

Analista$_G$:

- analizza i requisiti del progetto;
- studia il dominio applicativo del problema;
- redige l'*Analisi dei Requisiti*$_G$.

Progettista$_G$:

- progetta l'architettura del prodotto;
- prende decisioni tecniche e tecnologiche;
- redige la *Specifica Tecnica*.

Programmatore$_G$:

- scrive il codice del prodotto;
- implementa le funzionalità richieste;
- codifica le componenti dell'architettura del prodotto;
- redige il *Manuale Utente*.

Verificatore$_G$:

- verifica che il lavoro svolto sia conforme alle norme e alle specifiche tecniche del progetto;
- ricerca ed eventualmente segnala errori;
- redige il *Piano di Qualifica$_G$*.

4.1.2.4 Ticketing

Il gruppo *7Last* ha deciso di utilizzare ClickUp$_G$ come strumento di Issue Tracking System (ITS) per la gestione delle attività. L'amministratore$_G$ si occuperà di creare le attività e assegnarle ai membri del gruppo, i quali dovranno segnalarne lo stato di avanzamento. Su ClickUp$_G$ sarà possibile visualizzare le attività assegnate, i compiti da svolgere e le scadenze da rispettare tramite una dashboard$_G$ semplice ed intuitiva. Un'ulteriore funzionalità offerta da ClickUp$_G$ è quella del monitoraggio del tempo, strumento utile per il tracciamento del tempo impiegato per svolgere ciascuna attività. Altra caratteristica molto utile è l'integrazione con lo strumento *GitHub$_G$*, mediante la quale è possibile collegare le attività create su ClickUp$_G$ ai branch di GitHub$_G$ e alle relative pull request.

Apertura e ciclo di vita delle attività

In ClickUp$_G$ si possono creare diversi tipi di attività:

- **attività** (predefinito), attività generica corrispondente alle issue di altri ITS;
- **attività cardine**, attività principali che rappresentano le milestone$_G$ del progetto;
- **Product Backlog$_G$** Item (PBI), strumento chiave per pianificare il lavoro del team e assicurare che il prodotto evolva in modo coerente.

Le attività verranno create seguendo una struttura ben definita.

- **Tipo di attività**: indica il tipo di attività da svolgere.
- **Nome**: deve essere chiaro e conciso, in modo da identificare facilmente l'attività.
- **Assegnatari**: membri del team a cui viene assegnata l'attività.
- **Data di scadenza**: data entro la quale l'attività deve essere completata.

- **Priorità**: indica l'importanza dell'attività rispetto alle altre.
- **Stato**: indica lo stato di avanzamento dell'attività.
- **Tag**: etichetta che identifica l'attività in base alla sua tipologia.
- **Descrizione**: descrizione dettagliata dell'attività da svolgere.
- **GitHub**$_G$: collegamento con il branch di GitHub$_G$ relativo all'attività.

Ogni attività avrà un ciclo di vita ben definito, che prevede i seguenti stati:

- **da fare** indica che l'attività è stata assegnata ma non ancora iniziata;
- **in corso** specifica che il membro del team sta lavorando ad essa;
- **completata** evidenzia che l'attività è stata completata con successo e la pull request è stata accettata.

4.1.3 Metriche

Codice	Nome esteso
33M-RSI	Requirements Stability Index

Tabella 9: Metriche relative alla pianificazione

4.1.3.1 Strumenti

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

4.1.4 Coordinamento

4.1.4.1 Descrizione

La fase di coordinamento è un'attività cruciale per la gestione della comunicazione e delle risorse all'interno del progetto. Questa fase si concentra sulla creazione di un ambiente di lavoro collaborativo e armonioso, in cui i membri del team possano lavorare

insieme in modo efficace e produttivo. Il coordinamento coinvolge la pianificazione, l'organizzazione e il controllo delle attività, garantendo che ogni membro del team sia adeguatamente informato, motivato e allineato sui compiti da svolgere e sugli obiettivi da raggiungere.

4.1.4.2 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.

4.1.4.3 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.

4.1.4.4 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 *Sync Lab S.r.l.* viene scelto di adottare **Google Meet** per le comunicazioni sincrone esterne.

4.1.4.5 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$_G$. Qualora fosse necessario, il responsabile$_G$ provvederà a trovare un'altra data e un altro orario che possa andare bene a tutti i membri del gruppo, in alternativa gli eventuali assenti si informeranno sull'andamento della riunione tramite la lettura dei relativi verbali. Le riunioni interne saranno documentate e il verbale sarà redatto dal redattore della riunione. In queste riunioni i compiti del responsabile$_G$ 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.

4.1.4.6 Riunioni esterne

Durante lo svolgimento del progetto, è essenziale organizzare vari incontri con i committenti$_G$ o la proponente$_G$ 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$_G$, il quale è incaricato di pianificarli e agevolarne lo svolgimento in modo efficiente ed efficace. Sarà compito sempre del responsabile$_G$ anche l'esposizione dei punti di discussione alla proponente$_G$ o al committente$_G$, lasciando la parola ai membri del gruppo interessati se 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$_G$ assicurerà di informare tempestivamente la proponente$_G$ o i committenti$_G$, richiedendo eventualmente la possibilità di rinviare la riunione ad una data successiva.

Riunioni con l'azienda proponente$_G$

7Last si impegna ad organizzare incontri regolari con *Sync Lab 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$_G$ (SAL$_G$) 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 avvenute con la proponente$_G$ 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.

4.1.4.7 Strumenti

- **Discord**: impiegato per la comunicazione sincrona interna e asincrona esterna;
- **Telegram**: utilizzato per la comunicazione asincrona interna;
- **Google Meet**: adottato per la comunicazione sincrona esterna;
- **Gmail**: come servizio di posta elettronica.

4.2 Miglioramento

4.2.1 Introduzione

Secondo lo standard *ISO/IEC 12207:1995*, il miglioramento continuo è un processo che coinvolge l'identificazione e l'attuazione di azioni correttive e preventive per ottimizzare i processi e i prodotti. Questo processo è essenziale per garantire che il progetto sia completato con successo e in conformità con gli obiettivi e i requisiti stabiliti. Si compone di:

- **analisi dei dati**: questa fase coinvolge la raccolta e l'analisi dei dati pertinenti per identificare le aree che richiedono miglioramento;
- **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 processo 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 individuo del team dovrà compiere un processo di autoformazione per riuscire a svolgere al meglio il ruolo assegnato. La rotazione dei ruoli permetterà al nuovo occupante di un ruolo di apprendere le competenze necessarie da colui che lo ha precedentemente svolto, 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

Sync Lab S.r.l. mette a disposizione sessioni formative riguardo le tecnologie adottate nel progetto. Qualcosa sia necessario, verranno eseguite riunioni per chiarire eventuali dubbi.

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 inoltre 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**: include tutte le attività necessarie per ottenere un prodotto software o un servizio e comprende la preparazione delle richieste di offerta, la selezione del fornitore e la gestione del contratto;
- **fornitura**: include le attività eseguite per fornire un prodotto software o un servizio al cliente e include la preparazione della proposta, la negoziazione del contratto e la consegna del prodotto;
- **sviluppo**: copre tutte le attività di progettazione e sviluppo del software, dall'analisi dei requisiti$_G$ alla progettazione, implementazione, test e integrazione;
- **operatività**: riguarda le attività necessarie per utilizzare il sistema software in ambiente operativo e include la gestione delle operazioni quotidiane, il monitoraggio delle prestazioni, la gestione degli utenti, il backup e il ripristino dei dati, nonché il mantenimento della continuità operativa del sistema;
- **manutenzione**: include le attività necessarie per modificare il software dopo la sua consegna, per correggere difetti o migliorare prestazioni;

5.2.2 Processi di supporto

Forniscono il supporto necessario per i processi primari e comprendono:

- **documentazione**: gestione della documentazione necessaria per supportare le attività del ciclo di vita del software;
- **gestione della configurazione**: controllo delle modifiche al software per mantenere l'integrità e la tracciabilità delle configurazioni del software;
- **assicurazione della qualità**: garantisce che i prodotti e i processi del software soddisfino gli standard richiesti, include attività come la pianificazione della qualità, le revisioni di qualità, le ispezioni e le valutazioni per assicurare la conformità agli standard stabiliti;
- **verifica**: assicura che i prodotti del software soddisfino i requisiti specificati;

- **validazione**: assicura che il prodotto software soddisfi le esigenze e le aspettative dell'utente;
- **revisioni congiunte con il cliente**: fornisce una revisione periodica dei risultati del progetto con tutte le parti interessate;
- **audit**: verifica che i processi e i prodotti del software siano conformi ai requisiti, agli standard e alle procedure;
- **risoluzione dei problemi**: gestione e risoluzione dei problemi identificati nel software;

5.2.3 Processi organizzativi

Supportano l'organizzazione nel suo insieme e si compongono di:

- **gestione**: pianificazione, monitoraggio e controllo delle attività del progetto;
- **infrastrutture**: gestione dell'infrastruttura necessaria per supportare i processi di ciclo di vita del software;
- **miglioramento**: miglioramento continuo dei processi di ciclo di vita del software;
- **formazione**: pianificazione e fornitura della formazione necessaria per il personale coinvolto nei processi di ciclo di vita del software.

6 Metriche di qualità

La qualità di processo è un criterio fondamentale ed è alla base di ogni prodotto che rispecchi lo stato dell'arte. Per raggiungere tale obiettivo è necessario sfruttare delle pratiche rigorose che consentano lo svolgimento di ogni attività in maniera ottimale. Al fine di valutare nel miglior modo possibile la qualità del prodotto e l'efficacia dei processi, sono state definite delle metriche, meglio specificate di seguito. Il contenuto di questa sezione è necessario per identificare i parametri che le metriche devono rispettare per essere considerate accettabili o ottime. Esse sono state suddivise utilizzando lo *standard ISO/IEC 12207:1995*, il quale separa i processi di ciclo di vita del software, in tre categorie:

- processi di base e/o primari;
- processi di supporto;
- processi organizzativi.

6.1 Processi di base e/o primari

6.1.1 Fornitura

Nella fase di fornitura si definiscono le procedure e le risorse necessarie per la consegna del prodotto. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

1M-PV Planned Value

- **Definizione**: il *Planned Value* (o Valore Pianificato) rappresenta il valore del lavoro programmato per essere completato fino a un determinato momento. Si tratta del budget preventivato per lo sprint$_G$ in corso.
- **Come si calcola**:

  $$PV = BAC \times LP$$

  dove:

  - BAC: Budget At Completion;
  - LP: percentuale di lavoro pianificato.

- **Valore ammissibile**:
$$PV \geq 0$$
- **Valore ottimo**:
$$PV \leq BAC$$

2M-EV Earned Value

- **Definizione**: l'*Earned Value* (o Valore Guadagnato) rappresenta il valore del lavoro effettivamente completato fino al periodo in analisi.
- **Come si calcola**:
$$EV = BAC \times LC$$
dove:
  - BAC: Budget At Completion;
  - LC: percentuale di lavoro completato.
- **Valore ammissibile**:
$$EV \geq 0$$
- **Valore ottimo**:
$$EV \leq EAC\ (Estimated\ At\ Completion)$$

3M-AC Actual Cost

- **Definizione**: l'*Actual Cost* (o Costo Effettivo) rappresenta il costo effettivamente sostenuto per completare il lavoro fino al periodo in analisi.
- **Come si calcola**: si ottiene sommando tutti i costi effettivi sostenuti fino a quella data.
- **Valore ammissibile**:
$$AC \geq 0$$
- **Valore ottimo**:
$$AC \leq EAC$$

4M-SV Schedule Variance

- **Definizione**: la *Schedule Variance* (o Variazione di Programma) rappresenta la differenza tra il valore del lavoro effettivamente completato e il valore del lavoro pianificato, calcolata in percentuale.
- **Come si calcola**:
$$SV = \frac{EV - PV}{EV}$$
- **Valore ammissibile**:
$$SV \geq -10\%$$
- **Valore ottimo**:
$$SV \geq 0\%$$

5M-CV Cost Variance

- **Definizione**: la *Cost Variance* (o Variazione dei Costi) rappresenta la differenza tra il valore del lavoro effettivamente completato e il costo effettivamente sostenuto per completarlo, calcolata in percentuale.
- **Come si calcola**:
$$CV = \frac{EV - AC}{EV}$$
- **Valore ammissibile**:
$$CV \geq -10\%$$
- **Valore ottimo**:
$$CV \geq 0\%$$

6M-CPI Cost Performance Index

- **Definizione**: il *Cost Performance Index* rappresenta il rapporto tra il valore del lavoro effettivamente completato e i costi sostenuti per completarlo.
- **Come si calcola**:
$$CPI = \frac{EV}{AC}$$
- **Valore ammissibile**:
$$CPI \geq 0.8$$

- **Valore ottimo**:

$$CPI \geq 1$$

7M-SPI Schedule Performance Index

- **Definizione**: lo *Schedule Performance Index* rappresenta l'efficienza con cui il progetto sta rispettando il programma.
- **Come si calcola**:

$$SPI = \frac{EV}{PV}$$

- **Valore ammissibile**:

$$SPI \geq 0.8$$

- **Valore ottimo**:

$$SPI \geq 1$$

8M-EAC Estimate At Completion

- **Definizione**: l'*Estimate at Completion* (o Stima al Completamento) rappresenta una previsione aggiornata del costo totale del progetto basata sulle performance attuali, calcolata in base ai costi effettivamente sostenuti e ai costi stimati per completare il lavoro rimanente.
- **Come si calcola**:

$$EAC = AC + ETC$$

- **Valore ammissibile**:

$$EAC \leq BAC + 5\%$$

- **Valore ottimo**:

$$EAC \leq BAC$$

9M-ETC Estimate To Complete

- **Definizione**: l'*Estimate to Complete* (o Stima al Completamento) rappresenta una previsione del costo necessario per completare le attività rimanenti del progetto basata sulle performance attuali.

- **Come si calcola**:
$$ETC = EAC - AC$$
- **Valore ammissibile**:
$$ETC \geq 0$$
- **Valore ottimo**:
$$ETC \leq EAC$$

10M-OTDR On-Time Delivery Rate

- **Definizione**: l'*On-Time Delivery Rate* (o Tasso di Consegna nei Tempi) rappresenta la percentuale di attività completate entro la data di scadenza.
- **Come si calcola**:
$$OTDR = \frac{AP}{AT}$$
dove:
  - AP: attività completate entro la data di scadenza;
  - AT: attività totali.
- **Valore ammissibile**:
$$OTDR \geq 90\%$$
- **Valore ottimo**:
$$OTDR \geq 95\%$$

6.1.2 Sviluppo

Nella fase di sviluppo si realizza il prodotto software, seguendo le specifiche definite in fase di progettazione.

6.1.2.1 Analisi dei requisiti

Questa fase consiste nell'esaminare le richieste della proponente$_G$ e nel definire i requisiti che il prodotto dovrà soddisfare. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

11M-PRO Percentuale Requisiti Obbligatori

- **Definizione**: rappresenta la percentuale di requisiti obbligatori soddisfatti secondo quanto definito nel documento *Analisi dei Requisiti*$_G$.
- **Come si calcola**:
$$PRO = \frac{ROS}{ROT}$$
dove:
  - ROS: Requisiti Obbligatori Soddisfatti;
  - ROT: Requisiti Obbligatori Totali.
- **Valore ammissibile**:
$$PRO \geq 100\%$$
- **Valore ottimo**:
$$PRO = 100\%$$

12M-PRD Percentuale Requisiti Desiderabili

- **Definizione**: rappresenta la percentuale di requisiti desiderabili soddisfatti secondo quanto definito nel documento *Analisi dei Requisiti*$_G$.
- **Come si calcola**:
$$PRD = \frac{RDS}{RDT}$$
dove:
  - RDS: Requisiti Desiderabili Soddisfatti;
  - RDT: Requisiti Desiderabili Totali.
- **Valore ammissibile**:
$$PRD \geq 35\%$$
- **Valore ottimo**:
$$PRD = 100\%$$

13M-PRO Percentuale Requisiti Opzionali

- **Definizione**: rappresenta la percentuale di requisiti opzionali soddisfatti secondo quanto definito nel documento *Analisi dei Requisiti*$_G$.
- **Come si calcola**:
$$PRO = \frac{ROS}{ROT}$$
dove:
  - ROS: Requisiti Opzionali Soddisfatti;
  - ROT: Requisiti Opzionali Totali.
- **Valore ammissibile**:
$$PRO \geq 0\%$$
- **Valore ottimo**:
$$PRO \geq 100\%$$

6.1.2.2 Progettazione

Questa fase consiste nel definire l'architettura del prodotto software, in modo da soddisfare i requisiti definiti in fase di analisi. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

14M-PG Profondità delle Gerarchie

- **Definizione**: la profondità delle gerarchie rappresenta il numero massimo di livelli di annidamento delle classi.
- **Come si calcola**: si conta il numero massimo di livelli di annidamento delle classi.
- **Valore ammissibile**:
$$PG \leq 7$$
- **Valore ottimo**:
$$PG \leq 5$$

6.1.2.3 Codifica

Queste metriche aiutano a valutare la qualità del codice, la complessità e la manutenibilità.

15M-PPM Parametri Per Metodo

- **Definizione**: numero medio di parametri passati ai metodi. Un numero elevato di parametri può indicare che un metodo è troppo complesso o che potrebbe essere suddiviso in metodi più piccoli.
- **Come si calcola**:
$$PPM = \frac{P}{M}$$
dove:
  - P: numero totale di parametri;
  - M: numero totale di metodi.
- **Valore ammissibile**:
$$PPM \leq 7$$
- **Valore ottimo**:
$$PPM \leq 5$$

16M-CPC Campi Per Classe

- **Definizione**: numero medio di campi (variabili di istanza) per classe. Un numero elevato di campi dati può indicare che una classe sta facendo troppo e che potrebbe essere suddivisa in classi più piccole.
- **Come si calcola**:
$$CPC = \frac{CD}{CL}$$
dove:
  - CD: numero totale di campi dati;
  - CL: numero totale di classi.
- **Valore ammissibile**:
$$CPC \leq 8$$

- **Valore ottimo**:

$$CPC \leq 5$$

17M-LCPM Linee di Codice Per Metodo

- **Definizione**: numero medio di linee di codice per metodo. Metodi troppo lunghi possono essere difficili da leggere, capire e mantenere.
- **Come si calcola**:

$$LCPM = \frac{LC}{M}$$

  dove:

  - LC: numero totale di linee di codice;
  - M: numero totale di metodi.

- **Valore ammissibile**:

$$LCPM \leq 50$$

- **Valore ottimo**:

$$LCPM \leq 20$$

18M-CCM Complessità CicloMatica

- **Definizione**: la *Complessità CicloMatica* rappresenta la complessità di un programma sulla base del numero di percorsi lineari indipendenti attraverso il codice sorgente. Un valore elevato indica un codice più complesso e potenzialmente più difficile da mantenere.
- **Come si calcola**:

$$CCM = E - N + 2P$$

  dove:

  - E: numero di archi del grafo;
  - N: numero di nodi del grafo;
  - P: numero di componenti connesse.

- **Valore ammissibile**:

$$CCM \leq 6$$

- **Valore ottimo**:

$$CCM \leq 3$$

6.2 Processi di supporto

I processi di supporto si affiancano ai processi primari per garantire il corretto svolgimento delle attività.

6.2.1 Documentazione

La documentazione è un aspetto fondamentale per la comprensione del prodotto e per la sua manutenibilità. A livello pratico consiste nella redazione di manuali e documenti tecnici che descrivano il funzionamento del prodotto e le scelte progettuali adottate. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

19M-IG Indice Gulpease

- **Definizione**: l'Indice Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana. Misura la lunghezza delle parole e delle frasi rispetto al numero di lettere.
- **Come si calcola**:

$$IG = 89 + \frac{300 \times F - 10 \times L}{P}$$

  dove:

  - F: numero totale di frasi nel documento;
  - L: numero totale di lettere nel documento;
  - P: numero totale di parole nel documento.

- **Valore ammissibile**:

$$IG \geq 50\%$$

- **Valore ottimo**:

$$IG \geq 75\%$$

20M-CO Correttezza Ortografica

- **Definizione**: la correttezza ortografica indica la presenza di errori ortografici nei documenti.
- **Come si calcola**: si contano gli errori ortografici presenti nei documenti.
- **Valore ammissibile**:

$$0\ errori$$

- **Valore ottimo**:

$$0\ errori$$

6.2.2 Gestione della qualità

La gestione della qualità è un processo che si occupa di definire una metodologia per garantire la qualità del prodotto. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

21M-FU Facilità di Utilizzo

- **Definizione**: rappresenta il livello di usabilità del prodotto software mediante il numero di errori riscontrati durante l'utilizzo del prodotto da parte di un utente generico.
- **Come si calcola**: si contano gli errori riscontrati durante l'utilizzo del prodotto da parte di un utente che non ha conoscenze pregresse sul prodotto software.
- **Valore ammissibile**:

$$FU \leq 3\ errori$$

- **Valore ottimo**:

$$FU = 0\ errori$$

22M-TA Tempo di Apprendimento

- **Definizione**: indica il tempo massimo richiesto da parte di un utente generico per apprendere l'utilizzo del prodotto.
- **Come si calcola**: si misura il tempo necessario per apprendere l'utilizzo del prodotto da parte di un utente che non ha conoscenze pregresse sul prodotto software.

- **Valore ammissibile**:

$$TA \leq 12\ minuti$$

- **Valore ottimo**:

$$TA \leq 7\ minuti$$

23M-TR Tempo di Risposta

- **Definizione**: indica il tempo massimo di risposta del sistema sotto carico rilevato.
- **Come si calcola**: si misura il tempo massimo necessario per ottenere una risposta dal sistema.
- **Valore ammissibile**:

$$TR \leq 8\ s$$

- **Valore ottimo**:

$$TR \leq 4\ s$$

24M-TE Tempo di Elaborazione

- **Definizione**: indica il tempo massimo di elaborazione di un dato grezzo fino alla sua presentazione rilevato.
- **Come si calcola**: si misura il tempo massimo di elaborazione di un dato grezzo dal momento della sua comparsa nel sistema fino alla sua presentazione all'utente.
- **Valore ammissibile**:

$$TE \leq 10\ s$$

- **Valore ottimo**:

$$TE \leq 5\ s$$

25M-QMS Metriche di Qualità Soddisfatte

- **Definizione**: indica il numero di metriche implementate e soddisfatte, tra quelle definite.

- **Come si calcola**:
$$QMS = \frac{MS}{MT}$$
dove:
  - MS: metriche soddisfatte;
  - MT: metriche totali.
- **Valore ammissibile**:
$$QMS \geq 90\%$$
- **Valore ottimo**:
$$QMS = 100\%$$

6.2.3 Verifica

La verifica è un processo che si occupa di controllare che il prodotto soddisfi i requisiti stabiliti e sia pienamente funzionante. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

26M-CC Code Coverage

- **Definizione**: la *Code Coverage* indica quale percentuale del codice sorgente è stata eseguita durante i test. Serve per capire quanto del codice è stato verificato dai test automatizzati.
- **Come si calcola**:
$$CC = \frac{LE}{LT}$$
dove:
  - LE: linee di codice eseguite;
  - LT: linee di codice totali.
- **Valore ammissibile**:
$$CC \geq 80\%$$
- **Valore ottimo**:
$$CC = 100\%$$

27M-BC Branch Coverage

- **Definizione**: la *Branch Coverage* indica quale percentuale dei rami decisionali (percorsi derivanti da istruzioni condizionali come *if, for, while*) del codice è stata eseguita durante i test.
- **Come si calcola**:

$$BC = \frac{BE}{BT}$$

  dove:

  - BE: rami eseguiti;
  - BT: rami totali.

- **Valore ammissibile**:

$$BC \geq 80\%$$

- **Valore ottimo**:

$$BC = 100\%$$

28M-SC Statement Coverage

- **Definizione**: la *Statement Coverage* indica quale percentuale di istruzioni del codice è stata eseguita durante i test.
- **Come si calcola**:

$$SC = \frac{IE}{IT}$$

  dove:

  - IE: istruzioni eseguite;
  - IT: istruzioni totali.

- **Valore ammissibile**:

$$SC \geq 80\%$$

- **Valore ottimo**:

$$SC = 100\%$$

29M-FD Failure Density

- **Definizione**: la *Failure Density* indica il numero di difetti trovati in un software o in una parte di esso durante il ciclo di sviluppo rispetto alla dimensione del software stesso.
- **Come si calcola**:
$$FD = \frac{DF}{LT}$$
dove:
  - DF: difetti trovati;
  - LT: linee di codice totali.
- **Valore ammissibile**:
$$FD \leq 15\%$$
- **Valore ottimo**:
$$FD = 0\%$$

30M-PTCP Passed Test Case Percentage

- **Definizione**: la *Passed Test Case Percentage* indica la percentuale di test che sono stati eseguiti con successo su una base di test.
- **Come si calcola**:
$$PTCP = \frac{TS}{TT}$$
dove:
  - TS: test superati;
  - TT: test totali.
- **Valore ammissibile**:
$$PTCP \geq 90\%$$
- **Valore ottimo**:
$$PTCP = 100\%$$

6.2.4 Risoluzione dei problemi

La risoluzione dei problemi è un processo che mira a identificare, analizzare e risolvere le varie problematiche che possono emergere durante lo sviluppo. La gestione dei rischi, in particolare, si occupa di identificare, analizzare e gestire i rischi che possono insorgere durante lo svolgimento del progetto. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

31M-RMR Risk Mitigation Rate

- **Definizione**: la *Risk Mitigation Rate* indica la percentuale di rischi identificati che sono stati mitigati con successo.
- **Come si calcola**:
$$RMR = \frac{RM}{RT}$$
dove:
  - RM: rischi mitigati;
  - RT: rischi totali identificati.
- **Valore ammissibile**:
$$RMR \geq 80\%$$
- **Valore ottimo**:
$$RMR = 100\%$$

32M-NCR Rischi Non Calcolati

- **Definizione**: indica il numero di rischi occorsi che non sono stati preventivati durante l'analisi dei rischi.
- **Come si calcola**: si contano i rischi occorsi e non preventivati.
- **Valore ammissibile**:
$$NCR \leq 3$$
- **Valore ottimo**:
$$NCR = 0$$

6.3 Processi organizzativi

I processi organizzativi sono processi che si occupano di definire le linee guida e le procedure da seguire per garantire un'efficace gestione e coordinazione del progetto.

6.3.1 Pianificazione

La pianificazione è un processo che si occupa di definire le attività da svolgere e le risorse temporali e umane necessarie per il loro svolgimento. Per valutare la qualità di tale processo, sono state definite le seguenti metriche.

33M-RSI Requirements Stability Index

- **Definizione**: il *Requirements Stability Index* (RSI) indica la percentuale di requisiti che sono rimasti invariati rispetto al totale dei requisiti inizialmente definiti. Si tratta di una metrica utilizzata per misurare quanto i requisiti di un progetto rimangono stabili durante il ciclo di vita del progetto stesso, è particolarmente utile per comprendere l'impatto delle modifiche ai requisiti sul progetto.
- **Come si calcola**:
$$RSI = \frac{RI - (RA + RR + RC)}{RI}$$
dove:
  - RI: requisiti iniziali;
  - RA: requisiti aggiunti;
  - RR: requisiti rimossi;
  - RC: requisiti cambiati.
- **Valore ammissibile**:
$$RSI \geq 75\%$$
- **Valore ottimo**:
$$RSI = 100\%$$