Norme Di Progetto

2025-02-06

V1.0.0

sweetenteam@gmail.com

https://sweetenteam.github.io

Destinatari	Prof. Tullio Vardanega Prof. Riccardo Cardin
Redattori	Matteo Campagnaro Orlando Ferazzani
Verificatori	Matteo Campagnaro Orlando Ferazzani

Registro delle modifiche

Versione	Data	Autori	Verificatori	Dettaglio
1.0.0	2025-02-06	Matteo Campagnaro	Orlando Ferazzani	Approvazione per RTB
0.2.0	2024-12-03	Matteo Campagnaro	Orlando Ferazzani	Correzione capitolo: Processi Organizzativi
0.1.1	2024-12-03	Matteo Campagnaro	Orlando Ferazzani	Correzione capitolo: Processi di supporto - Gestione della configurazione - Sincronizzazione
0.1.0	2024-12-02	Matteo Campagnaro	Orlando Ferazzani	Stesura capitoli: Processi di supporto - Verifica e Validazione
0.0.11	2024-11-30	Matteo Campagnaro	Orlando Ferazzani	Stesura capitolo: Processi di supporto - Gestione della configurazione
0.0.10	2024-11-28	Matteo Campagnaro	Orlando Ferazzani	Stesura capitolo: Processi di supporto - Documentazione
0.0.9	2024-11-26	Orlando Ferazzani	Matteo Campagnaro	Stesura capitolo: Metriche di qualità del prodotto
0.0.8	2024-11-26	Matteo Campagnaro	Orlando Ferazzani	Stesura capitolo: Processi Primari - Svilippo
0.0.7	2024-11-26	Orlando Ferazzani	Matteo Campagnaro	Stesura capitolo: Metriche di Qualità del processo
0.0.6	2024-11-25	Orlando Ferazzani	Matteo Campagnaro	Stesura del capitolo: Processi Organizzativi - Standard ISO/IEC 9126
0.0.5	2024-11-22	Orlando Ferazzani	Matteo Campagnaro	Stesura del capitolo: Processi Organizzativi - Sottosezioni Infrastruttura, Miglioramento, Formazione
0.0.4	2024-11-22	Matteo Campagnaro	Orlando Ferazzani	Stesura del capitolo: Processi Primari - Fornitura
0.0.3	2024-11-21	Orlando Ferazzani	Matteo Campagnaro	Stesura del capitolo: Processi Organizzativi - Gestione dei Processi
0.0.2	2024-11-21	Matteo Campagnaro	Orlando Ferazzani	Stesura del capitolo: Introduzione
0.0.1	2024-11-20	Orlando Ferazzani	Matteo Campagnaro	Stesura del capitolo: Processi di Supporto - Gestione della qualità

Indice

Lista della immagini

1) Introduzione

1.1) Scopo del documento

Il presente documento ha lo scopo di definire le linee guida operative che il gruppo adotterà per la realizzazione del progetto didattico. In questo documento sono raccolte le procedure da seguire per ogni ***processo***$_G$ e attività. Per la realizzazione di questo documento è prevista la modifica e l'aggiunta di nuove indicazioni in base alle decisioni che verranno prese dal gruppo durante lo svolgimento del progetto. Tutti i membri del gruppo si impegnano a consultare regolarmente questo documento e ad attenersi scrupolosamente alle procedure in esso descritte, al fine di garantire un approccio professionale, coerente e uniforme nello svolgimento delle attività.

1.2) Scopo del progetto

Il team azzurrodigitale: utilizza quotidianamente diverse piattaforme per redigere documentazione e consultare informazioni essenziali per i progetti, questo può spesso comportare inefficienze. L'obiettivo del progetto Buddybot è lo sviluppo di una piccola piattaforma web con un'interfaccia ***chat***$_G$ per interagire con l'***IA***$_G$ che funga da assistente virtuale. Questo deve essere in grado di ottenere in modo facile e veloce informazioni dalle fonti specificate e di fornirle in base alle domande poste tramite ***chat***$_G$ in linguaggio naturale. Tali informazioni devono essere aggregate e centralizzate da diverse fonti tra cui ***GitHub***$_G$, ***Confluence***$_G$ e ***Jira***$_G$ permettendo un accesso facile e immediato con il fine di migliorare la produttività e dare supporto all'***OnBoarding***$_G$.

1.3) Glossario

Per garantire chiarezza e coerenza nella terminologia utilizzata nei documenti, è stato deciso di creare un glossario contenente le definizioni dei termini. Questo strumento raccoglierà tutti i termini specifici del dominio d'uso, accompagnati dai rispettivi significati. L'inclusione di un termine nel glossario sarà segnalata attraverso *questo stile*$_G$ dedicato.

1.4) Riferimenti

1.4.1) Riferimenti normativi

- `Norme di Progetto v1.0.0`
- Documentazione e presentazione del capitolato d'appalto C9: BuddyBot
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C9.pdf (Ultimo accesso: 2024-11-21)
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C9p.pdf (Ultimo accesso: 2024-11-21)
- Standard `ISO/IEC 12207:1995`
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf (Ultimo accesso: 2024-11-28)
- Regolamento del progetto didattico:
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf (Ultimo accesso: 2024-11-21)

1.4.2) Riferimenti informativi

- I processi di ciclo di vita del software
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T02.pdf (Ultimo accesso: 2024-11-21)
- `Glossario v1.0.0`:
  https://sweetenteam.github.io/docs/RTB/Documentazione_Interna/Glossario

- Documentazione *git*$_G$:
  https://git-scm.com/docs (Ultimo accesso: 2024-11-21)
- Documentazione *GitHub*$_G$:
  https://docs.github.com/en (Ultimo accesso: 2024-11-21)
- Documentazione *Typst*$_G$
  https://typst.app/docs/ (Ultimo accesso: 2024-11-21)

2) Processi primari

2.1) Fornitura

2.1.1) Scopo

Il processo primario di *fornitura*$_G$, definito dallo standard `ISO/IEC 12207:1995` (versione aggiornata al 2017), descrive l'insieme delle attività necessarie per sviluppare un prodotto software che soddisfi completamente i requisiti e le esigenze del *Committente*$_G$. Questo processo copre tutte le fasi, dalla formulazione della proposta fino alla consegna finale del prodotto, assicurando un percorso chiaro, strutturato, *efficiente*$_G$ ed *efficace*$_G$.
Lo scopo di questo processo è di tracciare e descrivere le attività svolte da ogni componente del gruppo SweeTenTeam, per determinare quanto lavoro è ancora da completare oppure è stato ultimato rispetto alle richieste del *proponente*$_G$.

2.1.2) Descrizione

Queste sono le attività definite nel processo primario di *fornitura*$_G$:

- **Acquisizione e preparazione**: si individuano le necessità del cliente e vengono definiti eventuali requisiti, con associate analisi dei costi pecuniari e temporali (preventivo), e valutazione delle varie opzioni;
- **Contrattazione**: vengono negoziati tra fornitore e cliente i termini e le condizioni contrattuali, e vengono stipulati di comune accordo obiettivi, costi, tempistiche e responsabilità di entrambe le parti;
- **Pianificazione**: vengono pianificate le attività e le stesure dei documenti utili alla realizzazione del progetto nel rispetto degli accordi presi;
- **Attuazione e controllo**: vengono eseguite le attività pianificate, controllando regolarmente e consistentemente lo stato di avanzamento e il rispetto degli impegni prefissati conformemente al costo, alle tempistiche e ai requisiti accordati in precedenza;
- **Revisione e valutazione**: vengono effettuate revisioni periodiche e confronti con il cliente, per assicurare il corretto svolgimento del progetto secondo i termini prefissati e risolvere dubbi e incertezze;
- **Completamento e consegna**: una volta completato il progetto, viene consegnato al cliente il prodotto finale, secondo quanto stipulato nel contratto.

2.1.3) Aspettative

Il gruppo SweeTen Team intende instaurare e mantenere uno stretto rapporto di collaborazione con l'azienda proponente azzurrodigitale: e, in particolare, con le figure di riferimento: Giorgio Vallini, Martina Daniele, Mattia Gottardello e Nicola Boscaro.

Grazie ad un dialogo continuo, il gruppo intende:

- Ricevere *feedback*$_G$ sul lavoro svolto;
- Verificare che i vincoli e i requisiti individuati corrispondano a quanto richiesto dal capitolato e dall'azienda proponente.

2.1.4) Rapporti con il proponente

L'azienda azzurrodigitale: ha messo a disposizione un canale di comunicazione costante e tempestivo per facilitare il dialogo con i responsabili del progetto assegnato al team. Questo canale, istituito sulla piattaforma *Discord*$_G$, consente di risolvere dubbi, ricevere consigli e gestire l'organizzazione del

lavoro. Oltre a *Discord*$_G$, per delle richieste più formali è possibile contattare la Proponente anche tramite posta elettronica.

Inoltre, è stato concordato di organizzare regolarmente riunioni di Stato Avanzamento Lavori (*SAL*$_G$), che si tengono generalmente ogni due settimane, il martedì alle 17:00 (con eventuali modifiche di data o orario in caso di necessità). Queste riunioni coincidono con la conclusione dello Sprint corrente e l'inizio del successivo. Durante gli incontri, il team presenta il lavoro svolto nello *Sprint*$_G$, riceve feedback dall'azienda, discute eventuali problemi riscontrati e pianifica le attività per il prossimo *Sprint*$_G$.

Per ogni colloquio con l'azienda proponente verrà redatto un resoconto sotto forma di Verbale Esterno, che riporterà nel nome e all'interno del documento la data del relativo incontro. I verbali redatti potranno essere consultati all'interno della relativa cartella presente sul *repository*$_G$ https://github.com/SweeTenTeam/Docs/tree/master/documents, disponibile per ogni *baseline*$_G$ del progetto.

2.1.5) Documentazione fornita

In aggiunta alle attività volte alla realizzazione del progetto, vengono redatti e resi disponibili all'azienda *proponente*$_G$ azzurrodigitale: e ai Committenti Prof. Vardanega e Prof. Cardin i seguenti documenti:

2.1.5.1) Valutazione dei Capitolati

La Valutazione dei Capitolati è un documento che fornisce una panoramica dettagliata sui capitolati d'appalto presentati il 15 ottobre 2023. Per ogni progetto vengono analizzate le richieste del *proponente*$_G$, individuate le possibili soluzioni e identificate eventuali criticità.

Ogni capitolato è stato suddiviso nelle seguenti sezioni:

- **Descrizione**: nome del progetto, azienda proponente, informazioni generali relative al prodotto da sviluppare secondo quanto descritto nella presentazione del capitolato;
- **Dominio applicativo**: contesto del progetto;
- **Dominio tecnologico**: tecnologie da utilizzare per lo sviluppo;
- **Aspetti positivi**;
- **Fattori critici**;
- **Conclusioni**: motivazioni sulla scelta/non scelta del capitolato.

2.1.5.2) Dichiarazione degli Impegni

Questo documento formalizza la volontà del gruppo di impegnarsi nella realizzazione del prodotto previsto da un capitolato specifico (nel nostro caso, *BuddyBot*$_G$, proposto dall'azienda azzurrodigitale:).

Oltre a dichiarare l'impegno, il documento include:

- La suddivisione del monte ore complessivo e dei singoli ruoli tra i membri del gruppo;
- La descrizione dettagliata dei ruoli e delle loro responsabilità, con riferimento alle specifiche del progetto;
- Un preventivo iniziale dei costi;
- La scadenza prefissata prevista per il completamento del progetto.

2.1.5.3) Glossario

Il Glossario è un documento di supporto pensato per i membri del team, i committenti e l'azienda *proponente*$_G$. La sua creazione ha l'obiettivo di eliminare ambiguità o incomprensioni legate alla terminologia utilizzata nella documentazione del progetto, garantendo chiarezza e coerenza nella comunicazione.

2.1.5.4) Lettera di presentazione

La Lettera di Presentazione è un documento che accompagna la documentazione e il prodotto forniti all'azienda ***proponente***$_G$ durante le fasi di revisione del progetto. Il suo scopo è fornire un rapido contesto sullo stato di avanzamento dei lavori (o sul loro avvio, nel caso della lettera relativa ai capitolati) e offrire una breve panoramica della documentazione prodotta fino a quel momento.

2.1.5.5) Analisi dei Requisiti

L'***Analisi dei Requisiti***$_G$ è un documento che definisce le funzionalità che il prodotto deve offrire e i requisiti da soddisfare affinché il ***software***$_G$ sviluppato risulti conforme alle richieste del proponente. Descrive in dettaglio i casi d'uso, i ***requisiti***$_G$ del progetto e le funzionalità attese per il prodotto finale, basandosi sugli obiettivi definiti. Inoltre, funge da base preliminare per la progettazione del software.

Contiene le seguenti informazioni:

- **Descrizione del prodotto**: obiettivo finale del prodotto e le sue funzionalità principali;
- **Lista dei casi d'uso**: identificazione di tutti gli scenari di utilizzo del sistema da parte degli utenti. Per ogni caso d'uso sono analizzati:
  - Scenario;
  - Attori coinvolti;
  - Azioni eseguibili.
- **Lista dei requisiti**: tutte le richieste o vincoli definiti dal proponente o dedotti dal team per la realizzazione del prodotto finale. I requisiti possono essere obbligatori, desiderabili e opzionali e verranno classificati dal gruppo a seconda della loro importanza.

2.1.5.6) Piano di Progetto

Il Piano di Progetto è un documento versionato e soggetto ad approvazione, redatto e aggiornato dal Responsabile del progetto con il supporto degli Amministratori durante l'intera durata del lavoro. Il suo scopo è delineare la pianificazione e la gestione delle attività necessarie per la realizzazione del progetto.

Il documento include le seguenti sezioni:

- **Analisi dei Rischi**: identifica le potenziali problematiche che potrebbero rallentare o ostacolare lo sviluppo. Per prevenire questi problemi, il gruppo propone soluzioni da applicare tempestivamente. I rischi sono classificati in:
  - ***Rischi organizzativi***$_G$;
  - ***Rischi tecnologici***$_G$.
- **Modello di sviluppo**: descrive l'approccio metodologico e strutturato adottato dal team per sviluppare il prodotto.
- **Pianificazione**: suddivide il progetto in periodi definiti, ciascuno corredato da eventi e attività specifiche. Per ogni periodo, il documento mostra una stima dell'impegno richiesto a ciascun membro del gruppo.
- **Preventivo**: fornisce una stima della durata di ogni periodo e il tempo necessario per completare tutte le attività previste.
- **Consuntivo**: analizza il lavoro effettivamente svolto rispetto al preventivo, valutando lo stato di avanzamento del progetto al termine di ciascun periodo.

2.1.5.7) Piano di Qualifica

Il Piano di Qualifica descrive le strategie di verifica e validazione per garantire la qualità del prodotto e dei processi del progetto. È un documento dinamico e aggiornato che definisce le pratiche di controllo qualità, con focus sulle metriche di valutazione del prodotto.

Le principali sezioni sono:

- **Qualità di processo**: garantisce che i processi di sviluppo siano ottimali, integrando la qualità in tutte le attività del ciclo di vita del *software*$_G$.
- **Qualità di prodotto**: assicura che il prodotto soddisfi i requisiti del progetto, concentrandosi su affidabilità, funzionalità, manutenibilità e usabilità.
- **Test**: include il piano di testing, comprendente test di unità, integrazione, sistema e accettazione, per verificare la correttezza finale del prodotto.

2.1.5.8) Norme di Progetto

Il documento Norme di Progetto ha lo scopo di definire un insieme di standard e regole riguardanti i processi e le loro modalità operative (*Way of Working*$_G$), da seguire obbligatoriamente dal team di sviluppo durante l'intero ciclo di vita del progetto, per garantire la qualità e la conformità agli obiettivi e requisiti stabiliti con il cliente.

Il documento è suddiviso nelle seguenti sezioni:

- Introduzione;
- Processi primari;
- Processi di supporto;
- Processi organizzativi;
- Standard `ISO/IEC 9126` per la qualità;
- Metriche di qualitá del processo;
- Metriche di qualitá del prodotto.

2.1.5.9) Manuale Utente

Il documento Manuale Utente serve a descrivere i requisiti minimi e le istruzioni utili all'installazione/utilizzo del prodotto finale. Descrive inoltre le funzionalità del prodotto e di come l'utente può usufruirne.

2.1.5.10) Specifica Tecnica

Il documento Specifica Tecnica ha lo scopo di descrivere e chiarire gli aspetti tecnici chiave del progetto, oltre a servire come guida per la codifica e la manutenzione del sistema. La sua finalità principale è fornire una descrizione dettagliata e approfondita dell'architettura implementativa del sistema, analizzando anche il codice e i design pattern utilizzati. Inoltre, il documento ha il compito di monitorare la copertura dei requisiti definiti nell'*Analisi dei Requisiti*$_G$.

2.1.5.11) Verbali Interni

La documentazione relativa alle riunioni interne avvenute tramite la piattaforma comunicativa *Discord*$_G$ del team, viene riportata sottoforma di verbali interni.

Lo scopo dei verbali interni è quello di fissare per iscritto:

- Discussioni, proposte, dubbi ed eventuali problemi riscontrati;
- Riassunto dell'andamento dell'ultimo periodo;
- Organizzazione per il prossimo periodo.

2.1.5.12) Verbali esterni

La documentazione relativa alle riunioni esterne avvenute tramite la piattaforma comunicativa *Google Meet*$_G$ dal team e dai referenti del progetto, viene riportata sottoforma di verbali esterni.

Lo scopo dei verbali esterni è quello di fissare per iscritto:

- Discussioni, proposte, dubbi ed eventuali problemi riscontrati;
- Resoconto del lavoro svolto durante l'ultimo Sprint con *feedback*$_G$ della Proponente;
- Organizzazione e obiettivi per il prossimo *Sprint*$_G$.

I verbali esterni, al contrario dei verbali interni, una volta redatti, verranno inviati telematicamente alla Proponente per una essere verificati e convalidati.

2.1.6) Strumenti

Gli strumenti software utilizzati nel processo di fornitura sono descritti di seguito.

2.1.6.1) *Telegram*$_G$

Servizio di messaggistica istantanea utilizzato per come metodo di comunicunicazione asincrona tra membri del gruppo.

2.1.6.2) *Gmail*$_G$

Servizio di posta elettronica utilizzato come metodo di comunicunicazione formale fra il team e i referenti del progetto.

2.1.6.3) *Discord*$_G$

Piattaforma utilizzata dal team per effettuare videochiamate e scambiare informazioni, file, dati utili, link e appunti. *Discord*$_G$, attraverso un apposito canale, viene utilizzato anche per avere un riscontro immediato con il team di azzurrodigitale:.

2.1.6.4) *Google Meet*$_G$

Piattaforma dove vengono svolti incontri, ogni due settimane, con l'azienda e i responsabili del progetto.

2.1.6.5) *Google Calendar*$_G$

Servizio che offre la possibilità di visualizzare le riunioni esterne e fissate dalla Proponente.

2.1.6.6) *Fogli Google*$_G$

Applicazione online che consente di creare e formattare fogli di calcolo. Viene utilizzata dal team per inserire e organizzare i dati relativi a preventivi e consuntivi dei vari periodi;

2.1.6.7) *GitHub*$_G$

Piattaforma online che consente agli sviluppatori di creare, salvare, gestire e condividere il proprio codice sorgente, facilitando la collaborazione e il controllo delle versioni.

2.1.6.8) *Typst*$_G$

Linguaggio utilizzato per la creazione e la redazione della documentazione.

2.2) Sviluppo

2.2.1) Introduzione

Lo sviluppo, delineato dallo standard `ISO/IEC 12207:1995`, si occupa di organizzare ed eseguire le attività per garantire che il prodotto completo soddisfi i tutti requisiti descritti nel ***contratto***$_G$. Questa sequenza include tutte le azioni intraprese dai membri del team, finalizzate alla realizzazione di un prodotto che rispecchi le esigenze e le specifiche definite dal ***Proponente***$_G$.

Le fasi che compongono lo sviluppo sono:

- **Analisi dei requisiti**;
- **Progettazione**;
- **Codifica**.

2.2.2) Analisi dei requisiti

2.2.2.1) Descrizione

L'***Analisi dei Requisiti***$_G$ definisce le funzionalità che il prodotto deve offrire e i requisiti da soddisfare affinché il ***software***$_G$ sviluppato risulti conforme alle richieste del ***proponente***$_G$. Descrive in dettaglio i casi d'uso, i ***requisiti***$_G$ del progetto e le funzionalità attese per il prodotto finale, basandosi sugli obiettivi definiti. Inoltre, funge da base preliminare per la progettazione del ***software***$_G$.

Contiene le seguenti informazioni:

- **Descrizione del prodotto**: obiettivo finale del prodotto e le sue funzionalità principali;
- **Lista dei casi d'uso**: identificazione di tutti gli scenari di utilizzo del sistema da parte degli utenti. Per ogni caso d'uso sono analizzati:
  - Scenario;
  - Attori coinvolti;
  - Azioni eseguibili.
- **Lista dei requisiti**: tutte le richieste o vincoli definiti dal proponente o dedotti dal team per la realizzazione del prodotto finale. I requisiti possono essere obbligatori, desiderabili e opzionali e verranno classificati dal gruppo a seconda della loro importanza.

2.2.2.2) Obiettivo

L'analisi mira a ottenere una comprensione dettagliata delle necessità degli utenti, delle caratteristiche del prodotto e delle condizioni operative previste.

Questa fase include tre aspetti principali:

- **Identificazione degli obiettivi e delle finalità del prodotto**: Definizione chiara e condivisa degli scopi e dei risultati attesi, tenendo conto delle esigenze degli utenti e delle proposte dell'azienda.
- **Supporto alla progettazione**: Fornitura di una base chiara e strutturata che consenta ai progettisti di definire in modo efficace l'architettura e il design del sistema, facilitando il lavoro successivo.
- **Miglioramento della comunicazione**: Creazione di un linguaggio comune che agevoli il dialogo tra i fornitori e gli ***stakeholders***$_G$, assicurando un allineamento completo sulle aspettative e sugli obiettivi.

2.2.2.3) Diagrammi *UML*$_G$ dei casi d'uso

Un diagramma di caso d'uso è uno strumento grafico e intuitivo utilizzato per rappresentare in modo sintetico e comprensibile le funzionalità di un sistema e le interazioni tra gli attori e il sistema stesso. Fornisce una visione d'insieme degli scenari d'uso, descrivendo le azioni necessarie per consentire agli utenti di completare specifiche attività. Questo tipo di diagramma è particolarmente utile durante la

fase di analisi e progettazione, in quanto evidenzia chiaramente i requisiti funzionali del sistema senza entrare nei dettagli implementativi.

Componenti principali:

- **Attore**: un'entità esterna (persona, organizzazione o sistema) che interagisce con il sistema per soddisfare un'esigenza.

Figura 1: Rappresentazione di un attore nel diagramma *UML*$_G$.

- **Caso d'uso**: una funzionalità offerta dal sistema con cui l'attore può interagire.

Figura 2: Rappresentazione di un caso d'uso nel diagramma *UML*$_G$.

- **Sottocasi**: descrivono in modo più dettagliato aspetti specifici di casi d'uso generali.

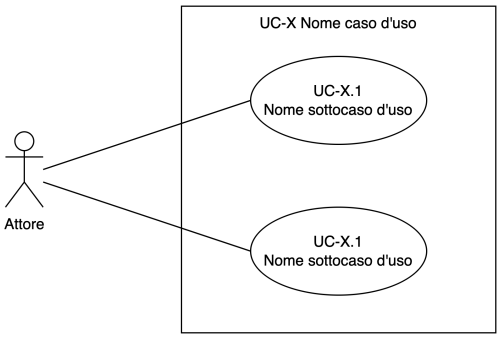

Figura 3: Rappresentazione di sottocasi d'uso nel diagramma *UML*$_G$.

- **Sistema**: rappresenta il software o prodotto in esame. I casi d'uso sono contenuti all'interno del sistema, mentre gli attori sono posizionati all'esterno.

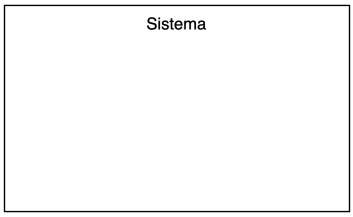

Figura 4: Rappresentazione di un sistema nel diagramma *UML*$_G$.

- **Relazione attore-caso d'uso**: indica come un attore utilizza una particolare funzionalità del sistema.

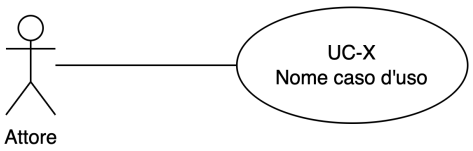

Figura 5: Rappresentazione di una relazione di associazione nel diagramma *UML*$_G$.

- **Generalizzazione tra attori**: descrive una relazione gerarchica in cui un attore specializzato eredita comportamenti e caratteristiche da un attore base (o «padre»). Questo consente di rappresentare differenze specifiche senza duplicare informazioni.

Figura 6: Rappresentazione di una generalizzazione tra attori nel diagramma *UML*$_G$.

Tipologie di relazioni tra casi d'uso:

- **Inclusione**: un caso d'uso (includente) integra un altro caso d'uso (incluso) come parte del suo processo, evitando ridondanze.

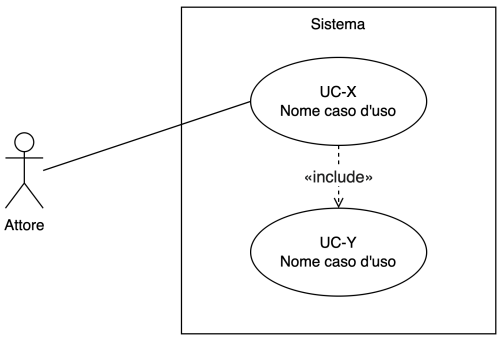

Figura 7: Rappresentazione di una relazione di inclusione nel diagramma *UML*$_G$.

- **Estensione**: un caso d'uso (estendibile) include opzionalmente un altro caso d'uso (esteso) al verificarsi di condizioni specifiche, gestendo scenari particolari senza complicare il caso principale.

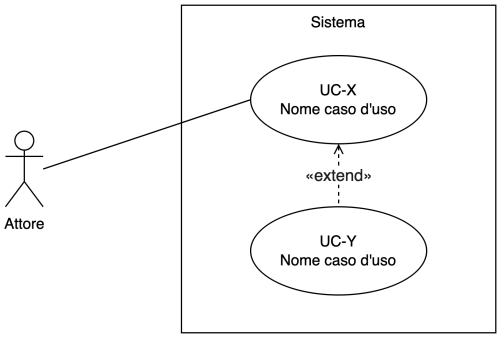

Figura 8: Rappresentazione di una relazione di estensione nel diagramma *UML*$_G$.

- **Generalizzazione**: un caso d'uso più specifico eredita comportamenti e proprietà da un caso d'uso più generale.

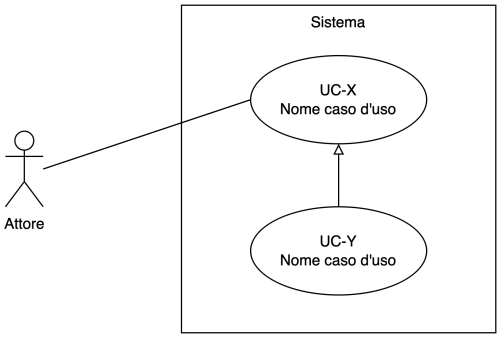

Figura 9: Rappresentazione di una relazione di generalizzazione nel diagramma *UML*$_G$.

2.2.2.4) Codice dei casi d'uso

A ogni caso d'uso è associato un codice univoco, strutturato nel seguente formato:

`UC-[X].[Y]`

Dove `X` rappresenta un identificativo univoco e `Y` indica una variante specifica del medesimo caso d'uso.

2.2.2.5) Requisiti

I requisiti di un *software*$_G$, stabiliti all'inizio del progetto con la *proponente*$_G$, stabiliscono in modo chiaro le prestazioni, le funzionalità, le caratteristiche e le restrizioni che il prodotto deve soddisfare. Questi requisiti fungono da guida nelle fasi di sviluppo, testing e valutazione finale, garantendo che il prodotto finito rispetti le aspettative e gli obiettivi fissati.

I requisiti possono essere suddivisi in queste categorie:

- **Funzionali**: Questi requisiti descrivono le funzionalità del sistema, cioè i servizi che il *software*$_G$ deve offrire per rispondere alle necessità dell'utente. Un requisito funzionale stabilisce come deve comportarsi il sistema.
- **Qualità**: Questi requisiti riguardano gli standard di qualità che il prodotto finale deve rispettare.
- **Vincolanti**: I requisiti di questo tipo indicano i limiti e le restrizioni stabilite nel capitolato che il prodotto deve seguire.
- **Prestazionali**: Questi requisiti determinano le performance che il sistema deve raggiungere.

2.2.2.6) Codice dei requisiti

A ciascun requisito è associato un codice univoco, strutturato nel seguente formato:

`R[X]-[Y]`

Dove `X` rappresenta la tipologia di requisito e `Y` è un identificativo progressivo.

2.2.3) Progettazione

2.2.3.1) Descrizione

L'obiettivo della progettazione è individuare una soluzione adeguata ai requisiti definiti nell'*Analisi dei Requisiti*$_G$. La progettazione del prodotto, a cura dei progettisti, si articola in due fasi principali:

Progettazione logica: Questa fase riguarda la selezione delle tecnologie, dei *framework*$_G$ e delle librerie da utilizzare per la realizzazione del prodotto, con l'obiettivo di garantirne l'adeguatezza attraverso il *Proof of Concept*$_G$ (PoC).

I principali aspetti di questa fase sono:

- Scelta delle tecnologie da utilizzare;
- Realizzazione del *Proof of Concept*$_G$ (PoC);
- Creazione dei diagrammi *UML*$_G$.

Progettazione di dettaglio: In questa fase si definisce l'architettura di base del prodotto, costruendo su quanto stabilito nella progettazione logica.

Gli elementi chiave includono:

- Diagrammi delle classi;
- Tracciamento delle classi;
- Applicazione dei *Design Pattern*$_G$
- *Test di unità*$_G$ per ciascun componente.

2.2.3.2) Vincoli di qualità

Le seguenti *best practice*$_G$ dovranno essere adottate durante la progettazione e lo sviluppo del prodotto:

- **Analisi**: La progettazione inizia solo dopo un'analisi approfondita dei requisiti concordati con la parte proponente.
- **Affidabilità**: Il *software*$_G$ è progettato per funzionare correttamente e in modo prevedibile.
- **Coesione**: Ogni componente è focalizzato su una singola responsabilità.
- **Chiarezza**: Il prodotto e la documentazione sono facilmente comprensibili, con diagrammi e spiegazioni.
- **Decoupling**: I componenti sono indipendenti, favorendo manutenibilità e flessibilità.
- **Flessibilità**: Il sistema si adatta facilmente a nuovi requisiti.
- **Incapsulamento**: I dettagli interni sono protetti e l'interazione avviene solo tramite interfacce stabilite.
- **Modularità**: L'architettura è divisa in parti autonome, facilmente gestibili.
- **Riusabilità**: I moduli e il codice sono creati per essere utilizzati in diversi contesti.
- **Scalabilità**: Il sistema può gestire crescenti carichi e nuove funzionalità senza problemi.
- **Semplicità**: La progettazione è pulita, senza complessità inutili.
- **Sufficienza**: Si assicura che tutti i requisiti stabiliti vengano soddisfatti.

2.2.3.3) Diagrammi delle classi

I diagrammi delle classi sono utilizzati per completare la progettazione del prodotto, poiché sono fondamentali per definire le caratteristiche e le relazioni tra i vari componenti del sistema.

Una classe è rappresentata graficamente come un rettangolo suddiviso in tre sezioni principali:

1. **Nome della classe**: Questo campo rappresenta il nome identificativo della classe, che viene scritto in grassetto seguendo la convenzione *PascalCase*$_G$, eccetto per le classi astratte, il cui nome è scritto in corsivo.
2. **Attributi**: Gli attributi sono i campi dati della classe e sono rappresentati singolarmente, ciascuno su una riga separata. Il formato di rappresentazione degli attributi è il seguente:

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

   Dove:

   - **Visibilità**: Indica l'accessibilità dell'attributo e può essere:
     - **-** : visibilità privata.

  - **#** : visibilità protetta.
  - **+** : visibilità pubblica.
  - **~** : visibilità di package.
- **Nome**: Il nome dell'attributo, che deve essere unico per ogni campo. La convenzione di nomenclatura è *camelCase*$_G$ mentre per le costanti, i nomi sono scritti in maiuscolo.
- **Tipo**: Specifica il tipo dell'attributo, che può essere primitivo o un tipo complesso.
- **<T>**: Nel caso di collezioni o container (come liste o array), si indica il tipo di oggetto contenuto.
- **Molteplicità**: Indica la quantità di elementi nel caso di collezioni. Se la dimensione è sconosciuta, si usa **"[*]"**. Se si tratta di un singolo elemento, la molteplicità è opzionale.
- **Valore/i di default**: Ogni campo può essere inizializzato con un valore predefinito, se applicabile.

3. **Firme dei metodi**: Le firme dei metodi descrivono il comportamento di ciascun metodo e sono rappresentate ciascuna su una riga separata. Il formato di rappresentazione è il seguente:

   Visibilità Nome (Parametri Formali): Tipo di ritorno

   Dove:

   - **Visibilità**: Indica l'accessibilità del metodo e segue gli stessi simboli degli attributi:
     - **-** : visibilità privata.
     - **#** : visibilità protetta.
     - **+** : visibilità pubblica.
     - **~** : visibilità di package.
   - **Nome**: Il nome del metodo, che deve essere descrittivo e indicare chiaramente la funzionalità del metodo stesso.
   - **Parametri formali**: I parametri richiesti dal metodo, separati da virgole.
   - **Tipo di ritorno**: Il tipo di valore che il metodo restituisce, se applicabile.

Per convenzione, i metodi di tipo getter/setter e i costruttori non vengono generalmente rappresentati nel diagramma. Se un attributo o un metodo è assente, questo viene indicato graficamente, e nel caso di ereditarietà, eventuali metodi ereditati che non sono specificati possono essere ignorati.

2.2.3.4) Relazioni tra classi

Le classi in un diagramma delle classi possono essere collegate tramite diverse tipologie di relazioni, ognuna rappresentata graficamente in modo specifico.

Ecco le principali:

- **Dipendenza**: Una classe dipende da un'altra quando utilizza i suoi metodi o attributi. È rappresentata con una freccia tratteggiata.

  Può essere etichettata con:

  - «**use**»: quando la classe utilizza metodi o attributi dell'altra classe.
  - «**create**»: quando crea un'istanza prima di utilizzarla.

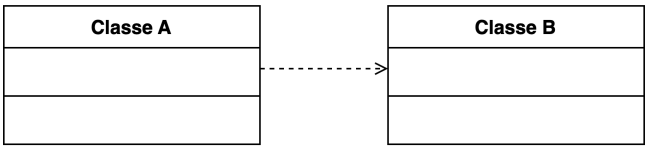

Figura 10: Rappresentazione di una dipendenza.

- **Associazione**: Quando una classe contiene un riferimento ad un'altra, indicata con una linea continua. La molteplicità (quanti oggetti di una classe sono associati a un'altra) può essere espressa vicino alla freccia, e se è 1, può essere omessa.

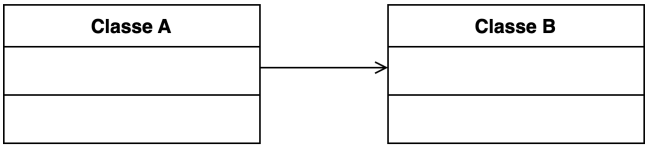

Figura 11: Rappresentazione di una associazione.

- **Aggregazione**: Una classe è composta da un'altra, ma le classi componenti possono esistere indipendentemente. È rappresentata con una freccia con un diamante vuoto.

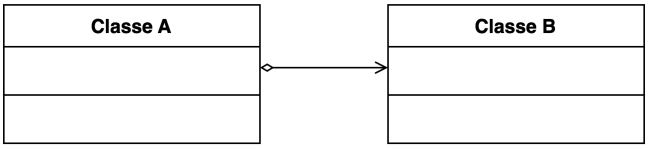

Figura 12: Rappresentazione di una aggregazione.

- **Composizione**: Una relazione più forte rispetto all'aggregazione, dove le classi «parti» non possono esistere senza la classe principale. È indicata con una freccia con un diamante pieno.

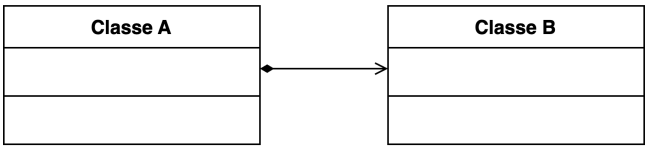

Figura 13: Rappresentazione di una composizione.

- **Generalizzazione**: Quando una sottoclasse eredita da una superclasse, acquisendo i suoi attributi e metodi. È rappresentata con una freccia continua e vuota.

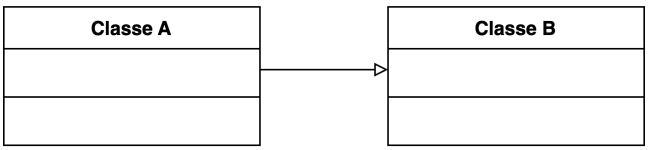

Figura 14: Rappresentazione di una generalizzazione.

- **Interface Realization**: Una classe concreta implementa i metodi definiti in un'interfaccia. È collegata all'interfaccia tramite una linea semplice, con l'interfaccia rappresentata da un cerchio.

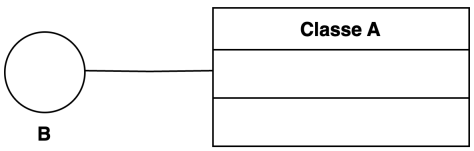

Figura 15: Rappresentazione di una realizzazione di interfaccia.

2.2.3.5) Design Pattern

Un *design pattern*$_G$ è una soluzione consolidata, ampiamente riconosciuta per risolvere problemi che tendono a ripresentarsi frequentemente durante il processo di sviluppo *software*$_G$. Si tratta di un approccio testato che offre un metodo chiaro e strutturato per affrontare situazioni ricorrenti, facilmente riutilizzabile in diverse circostanze. Questi pattern derivano dall'esperienza pratica e sono progettati per aumentare l'efficienza, migliorare la qualità e favorire una maggiore manutenibilità del *software*$_G$. Ogni pattern include una descrizione approfondita del suo scopo nell'architettura, contribuendo così a comprendere il suo valore e il modo migliore per applicarlo.

2.2.3.6) Test

Il processo di test ha come scopo quello di garantire la qualità del prodotto in tutte le sue fasi, dalla progettazione e sviluppo fino alla fase finale. Durante la definizione dei test, vengono stabiliti i requisiti necessari, i casi da testare e i criteri di accettazione per convalidare il *software*$_G$. L'obiettivo primario è quello di scoprire e risolvere errori, *bug*$_G$ e difetti nel *software*$_G$ prima della sua distribuzione. Inoltre, il testing è essenziale per garantire che il prodotto rispetti appieno le specifiche e i requisiti definiti con il cliente.

2.2.4) Codifica

L'obiettivo della codifica, che spetta ai programmatori, è tradurre le specifiche in un prodotto funzionante e utilizzabile.

Questo prodotto deve poi rispondere a vari requisiti, tra cui:

- *Test di unità*$_G$;
- **Conformità all'uso** dei metodi di codifica e agli standard adottati;
- **Test di integrazione**;
- **Funzionalità e manutenibilità** del prodotto finale.

2.2.4.1) Stile di Codifica

Le seguenti norme, ispirate ai principi di `Clean Code di Robert C. Martin`, dovranno essere rispettate dal team durante la scrittura del codice del prodotto:

- **Chiarezza e leggibilità**:
  - Il codice deve essere chiaro e facilmente comprensibile. Se necessario, migliorare i nomi e aggiungere commenti esplicativi.
  - Ogni parte del codice deve rispondere chiaramente alla domanda «Cosa fa?», rendendolo comprensibile nel tempo.
- **Nomina delle variabili, metodi e classi**:
  - Usare *PascalCase*$_G$ per le classi e *camelCase*$_G$ per variabili e metodi. I nomi devono essere chiari, descrittivi e concisi.
  - Evitare abbreviazioni poco chiare.
- **Lunghezza del metodo e della classe**:
  - Ogni metodo deve implementare una sola funzione e non superare le 20-30 righe di codice.
  - Ogni classe deve essere chiara, facilmente comprensibile e gestire un'unica responsabilità.
- **Indentazione e formattazione**:
  - Usare 4 spazi per l'indentazione.
  - Separare i blocchi di codice con righe vuote per migliorarne la leggibilità.
- **Commenti e documentazione**:
  - Scrivere commenti solo quando il codice non è autoesplicativo. I commenti devono spiegare il «perché» di una decisione, non il «come».

  - I commenti devono essere chiari, concisi e non ridondanti.
  - Ogni classe e metodo deve avere una documentazione che ne descriva la funzionalità e i parametri.
- **Gestione degli errori**:
  - Gli errori vanno gestiti esplicitamente tramite eccezioni.
  - Utilizzare eccezioni per gli errori inaspettati e mantenere il flusso di lavoro stabile.
- **Testabilità e unit test**:
  - Ogni componente deve essere facilmente testabile.
  - Scrivere unit test per metodi complessi o che interagiscono con componenti esterni.
  - Ogni test deve essere indipendente e focalizzarsi su un comportamento specifico.
- **Controllo dei cicli annidati**:
  - Limitare i cicli annidati a 3 livelli.
  - Usare funzioni di ordine superiore per semplificare il flusso logico.
- **Modularità e riusabilità**:
  - Il codice deve essere suddiviso in moduli riutilizzabili e facilmente testabili.
  - Evitare duplicazioni, usando funzioni e metodi per centralizzare la logica.
- **Performance e ottimizzazione**:
  - Scrivere codice leggibile prima di ottimizzare le performance.
  - Ottimizzare solo quando la performance è effettivamente un problema misurato.

3) Processi di supporto

3.1) Documentazione

3.1.1) Scopo

La documentazione è importante nello sviluppo del *software*$_G$ perché aiuta a tenere traccia di tutte le attività e le decisioni del progetto. Contiene informazioni utili e direttive per chi sviluppa, utilizza o distribuisce il *software*$_G$. È un supporto pratico per il team, che serve a organizzare meglio il lavoro in ogni fase. Aggiornandola regolarmente, rimane chiara e utile, rendendo più facile la manutenzione del progetto.

3.1.2) Ciclo di vita

Le sette fasi principali che compongono il ciclo di vita di un documento sono:

1. **Creazione o adattamento del template**: la prima fase prevede la creazione o l'adattamento di un modello per il documento corrente. Questo modello definisce struttura, formattazione e informazioni base, come titolo, autore e data.
2. **Pianificazione e assegnazione delle sezioni**: in questa fase le sezioni del documento vengono pianificate e assegnate ai redattori incaricati. I redattori sono responsabili della stesura delle proprie sezioni in conformità con le norme di progetto.
3. **Raccolta dei contenuti e preparazione della prima bozza**: i redattori raccolgono i materiali necessari e preparano una bozza iniziale del documento, che sarà la base per successive discussioni e revisioni.
4. **Stesura delle sezioni**: i redattori completano le sezioni assegnate, rispettando il modello e le linee guida del progetto.
5. **Verifica dei contenuti**: i redattori controllano che i contenuti delle sezioni siano coerenti con le norme di progetto e privi di errori.
6. **Revisione**: un verificatore revisiona l'intero documento per accertarsi che non ci siano errori o mancanze.
7. **Approvazione e pubblicazione**: il documento viene approvato da un responsabile e pubblicato nella sua versione definitiva.

3.1.3) Documentation as Code

Per la documentazione viene adottato l'approccio *Documentation as Code*$_G$, che si basa sull'utilizzo di pratiche e strumenti tipici dello sviluppo *software*$_G$ per la creazione, gestione e distribuzione dei documenti. Questo metodo offre un modello organizzato e flessibile, con i seguenti punti chiave:

- **Automazione**: sfruttare strumenti automatizzati per generare e aggiornare i documenti.
- **Collaborazione**: promuovere il lavoro di squadra attraverso piattaforme e strumenti condivisi.
- **Distribuzione**: rendere i documenti facilmente accessibili a tutti gli interessati.
- **Integrazione incrementale**: incorporare modifiche in modo graduale e continuo.
- **Versionamento**: mantenere traccia di tutte le modifiche per garantire trasparenza e controllo.

3.1.4) Sistema di composizione tipografica

Per la creazione tipografica dei documenti, abbiamo scelto di utilizzare *Typst*$_G$ anziché il tradizionale *LaTeX*$_G$.

Typst$_G$ presenta diversi vantaggi rispetto a *LaTeX*$_G$:

- **Facilità d'uso**: la sintassi è simile a quella di `Markdown`, rendendo l'approccio più semplice.

- **Programmabilità vera e propria**: al posto di un sistema basato su macro, offre una Programmabilità più diretta e flessibile.
- **Compilazione rapida**: la generazione dei documenti avviene in tempi quasi immediati.

L'adozione di *Typst*$_G$ semplifica la creazione e la manutenzione della documentazione, permettendo ai redattori di concentrarsi sul contenuto piuttosto che sulla parte grafica, e assicurando al contempo coerenza nella documentazione del progetto.

Il template sviluppato e utilizzato per questo scopo è visionabile all'interno della repository Docs.

3.1.5) Struttura dei documenti

3.1.5.1) Intestazione

La prima pagina funge da intestazione del documento e presenta gli elementi elencati di seguito:

- **Logo del gruppo**: presente nel percorso `images/logos/sweetenteam.png`;
- **Nome del documento**;
- **Data**: la data in cui è stata approvata l'ultima versione del documento;
- **Versione**: la versione corrente del documento;
- **Email**: `sweetenteam@gmail.com`;
- **Link al sito vetrina**: https://sweetenteam.github.io
- **Logo dell'Università di Padova**: presente nel percorso `images/logos/Universita_Padova_transparent.png`;
- **Destinatari**: a chi è il rivolto il documento;
- **Redattori**: incaricati della stesura del documento;
- **Verificatori**: incaricati della verifica del documento.

3.1.5.2) Registro delle modifiche

La seconda pagina è dedicata al ***registro delle modifiche*** che è organizzato in una tabella che consente di monitorare i cambiamenti apportati al documento.

La tabella include i seguenti campi:

- **Versione**: il numero di versione del documento;
- **Data**: la data di approvazione della versione del documento;
- **Autori**: le persone che hanno effettuato le modifiche;
- **Verificatori**: le persone che hanno approvato le modifiche;
- **Dettaglio**: una breve descrizione delle modifiche effettuate.

3.1.5.3) Indice

Nella pagina che segue il registro delle modifiche è presente l'***indice*** dove vengono elencate le sezioni contenute nel documento.

3.1.5.4) Lista delle immagini e delle tabelle

Nella pagina che segue l'indice, qualora nel documento siano presenti delle immagini o delle tabelle, sono presenti la ***lista delle immagini*** e la ***lista delle tabelle*** che descrivono il contenuto e la posizione nel documento di questi elementi.

3.1.5.5) Corpo del documento

Il contenuto del documento è suddiviso in capitoli, ognuno dei quali è formato da più sezioni ed eventuali sottosezioni.

3.1.5.6) Corpo dei verbali interni

Il contenuto del verbale interni è suddiviso nelle seguenti sezioni:

- **Contenuti del verbale**:
  - **Informazioni sulla riunione**:
    - Luogo;
    - Ora di inizio;
    - Ora di fine;
    - Partecipanti;
  - **Ordine del giorno**: un elenco di ciò che verrà discusso durante la riunione;
- **Decisioni prese**: sezione che elenca in forma testuale le decisioni prese durante l'incontro. Alcune di queste potrebbero risultare in "attività individuate".
- **Prossime attività**: illustrazione dettagliata delle attività assegnate ai diversi membri del gruppo a conclusione dell'incontro. Queste informazioni, inserite in un'apposita tabella, riportano:
  - **ID**: collegamento alla relativa issue su GitHub
  - **Descrizione**: breve spiegazione dell'attività;
  - **Assegnatari**: i nomi degli incaricati a svolgere l'attività.

3.1.5.7) Corpo dei verbali esterni

Il contenuto del verbale esterni è suddiviso nelle seguenti sezioni:

- **Contenuti del verbale**:
  - **Informazioni sulla riunione**:
    - Luogo;
    - Ora di inizio;
    - Ora di fine;
    - Partecipanti;
    - Partecipanti esterni.
  - **Ordine del giorno**: un elenco di ciò che verrà discusso durante la riunione;
- **Sintesi dell'incontro**: un elenco di ciò che verrà discusso durante la riunione;
- **Domande effettuate e relative risposte**;
- **Prossimi step**: prossimi incontri e attività generali su cui riflettere e organizzarsi;
- **Decisioni prese**: sezione che elenca in forma testuale le decisioni prese durante l'incontro. Alcune di queste potrebbero risultare in "attività individuate".

3.1.5.8) Documenti del progetto

Verranno redatti i seguenti documenti:

- **`Dichiarazione Impegni`**;
- **`Valutazione Capitolati`**;
- **`Verbali Interni`**;
- **`Verbali Esterni`**;
- **`Lettera di Presentatione`**.
- **`Norme di Progetto`**;
- **`Piano di Progetto`**;

- **`Piano di Qualifica`**;
- **`Analisi dei Requisiti`**;
- **`Glossario`**;
- **`Specifica Tecnica`**;
- **`Manuale Utente`**;

3.1.6) Regole stilistiche

3.1.6.1) Nomi assegnati ai file

I documenti PDF presenti nella repository Docs, seguono precise convenzioni di denominazione in base al tipo di documento.

- **Verbali**: per i verbali viene utilizzata una nomenclatura del tipo:

  `[Verbale_Interno]/[Verbale_Esterno] + _ + [Data] + _ + [Versione]`

  - **Spaziatura**: fra le parole la spaziatura verrà sostituita da un underscore, («_»);
  - **Data**: scritta nel formato **`YYYY-MM-DD`**, dove `YYYY` indica l'anno, `MM` il mese e `DD` il giorno
  - **Versione**: scritta nel formato: **`v.X.Y.Z`**.

- **Altri documenti**: i nomi di tutti gli altri documenti presenti nel progetto invece saranno del tipo:

  `[Nome_del_File] + _ + [Versione]`

  - **Nome_del_File**: le parole sono scritte in Title Case (con la prima lettera maiuscola per le parole principali, come sostantivi, verbi, aggettivi e avverbi, e in minuscolo per le parole secondarie, come preposizioni, congiunzioni e articoli) e separate da un underscore, («_»).
  - **Spaziatura**: fra le parole la spaziatura verrà sostituita da un underscore, («_»);
  - **Versione**: scritta nel formato: **`v.X.Y.Z`**.

I file sorgente `.typ` non includono la versione nel nome. La versione viene aggiunta ai PDF automaticamente durante la compilazione. Usare lo stesso nome per i documenti permette di tracciare facilmente le modifiche con Git, sfruttando la funzione «**`diff`**» per confrontare i cambiamenti nei file di testo.

3.1.6.2) Stile del testo

Nei documenti è possibile utilizzare diversi stili di formattazione, tra cui il *corsivo*, il **grassetto** e il `monospace`, a discrezione del redattore. Questi strumenti sono impiegati per migliorare la leggibilità, enfatizzare concetti chiave e rendere il contenuto più chiaro e accessibile ai lettori. Per gli stessi scopi, possono essere utilizzati ulteriori elementi di impaginazione, come interruzioni di riga e di pagina, varie modalità di allineamento del testo e, quando ritenuto opportuno, il logo della proponente «azzurrodigitale:» in sostituzione del testo «**`AzzurroDigitale`**».

3.1.6.3) Elenchi puntati

Ogni elemento dell'elenco inizia con la lettera maiuscola e termina con un punto e virgola («**`;`**»), tranne l'ultimo, che si chiude con un punto («**`.`**»), a meno che non contenga più frasi: in tal caso può terminare con un punto indipendentemente dalla posizione. Il grassetto viene utilizzato solo dove appropriato.

3.1.6.4) Formato delle date

Per le date è stato deciso di adottare lo standard internazionale `ISO 8601`, nella forma **`YYYY-MM-DD`**, dove:

- **`YYYY`**: l'anno con 4 cifre;
- **`MM`**: il mese con 2 cifre;

- **DD**: il giorno con 2 cifre.

3.1.6.5) Descrizione immagini

Ogni immagine o tabella deve presentare una descrizione associata, utile a fornire una breve descrizione o spiegazione del contenuto.

3.1.7) Strumenti

Per la realizzazione della documentazione sono stati scelti dal team i seguenti strumenti:

- *Typst*$_G$: linguaggio utilizzato per la stesura dei documenti;
- *GitHub*$_G$: piattaforma per l'hosting di codice sorgente, utilizzata per condividere file, verificarli, gestirne le versioni e automatizzare determinati processi, seguendo il principio della *Documentation as Code*$_G$.

3.2) Gestione della configurazione

3.2.1) Descrizione e Scopo

Il processo di gestione della configurazione si applica durante l'intero ciclo di vita di un progetto *software*$_G$ e definisce regole per assicurare il monitoraggio delle modifiche apportate a documentazione e codice. Questo processo permette di accedere in qualsiasi momento allo storico delle modifiche, offrendo una visione chiara delle motivazioni alla base di ogni cambiamento e dei relativi autori, migliorando così la tracciabilità e la responsabilità all'interno del team.

3.2.2) Versionamento

Il versionamento consente di monitorare e documentare le modifiche apportate a un file nel tempo, permettendo di visualizzare l'evoluzione del contenuto e, se necessario, ripristinare una versione precedente.

Il team ha adottato un formato specifico per il versionamento dei documenti: **X.Y.Z**, dove ciascun valore rappresenta:

- **X**: identifica il completamento di una fase significativa del progetto. Questa versione viene definita dal responsabile durante la validazione.
- **Y**: indica una versione intermedia che integra più modifiche incrementali (**Z**) in modo coerente. Le versioni con il formato **X.Y.0** sono considerate stabili, anche se non completamente definitive.
- **Z**: corrisponde a modifiche incrementali minori, come l'aggiunta o l'aggiornamento di una sezione.

Ogni modifica approvata comporta un incremento di versione, il cui impatto dipende dall'entità della modifica stessa. Inoltre, un aggiornamento di una cifra comporta l'azzeramento delle cifre a destra di essa. Per i verbali e il glossario, viene utilizzato un formato ridotto a due cifre, poiché, per i primi, il testo è generalmente breve e rapido da redigere, per il glossario, le modifiche devono sempre portare a una versione stabile.

3.2.3) Tecnologie utilizzate

- **Git**: strumento per il controllo di versione, utilizzato per tracciare le modifiche a documenti e codice sorgente.
- **GitHub**: servizio di hosting per progetti software utilizzato per gestire il versionamento e coordinare le attività del team. Include anche un *Issue Tracking System*$_G$ per monitorare e gestire segnalazioni e attività.

3.2.4) Repository

3.2.4.1) Lista repository

Il team all'interno della propria *GitHub Organization*$_G$ utilizza 3 repository:

- **`Docs`**: repository destinata alla gestione della documentazione del progetto;
- **`BuddyBot`**: repository dedicata allo sviluppo e all'implementazione del software;
- **`SweeTenTeam.github.io`**: repository utilizzata per il sito vetrina del progetto.

3.2.4.2) Struttura della repository Docs

Il contenuto della repository è organizzato in due *branch*$_G$ distinti, ciascuno con una funzione specifica:

- **`master`**: contiene i file PDF compilati.
- **`develop`**: ospita i sorgenti *Typst*$_G$. Ogni volta che un file viene aggiunto o modificato in questo *branch*$_G$, il sistema avvia automaticamente la compilazione e carica il risultato nel branch `master`, rendendo i documenti compilati accessibili a tutti.

Per entrambi i *branch*$_G$, è stata adottata una struttura chiara che separa i documenti dai file di supporto, come `LICENSE`, `README`, *actions*, *template* e *images*. Infatti la documentazione è raccolta all'interno della cartella **`documents`**, che include i materiali relativi alle varie fasi del progetto:

- **`1-Candidatura`**: documenti relativi alla gara per l'aggiudicazione dell'appalto dei capitolati.
- **`2-RTB`**: *Requirements and Technology Baseline*, definisce i requisiti da soddisfare in collaborazione con il proponente e giustifica la scelta di tecnologie, framework e librerie, dimostrandone l'adeguatezza e la compatibilità. Questa cartella contiene tutti i documenti relativi alla prima revisione di avanzamento (**`RTB`**).
- **`3-PB`**: *Product Baseline*, valuta la maturità della baseline architetturale del software e la sua implementazione, con l'obiettivo di raggiungere l'*MVP*$_G$ (Minimum Viable Product). Questa cartella contiene tutti i documenti relativi alla seconda e ultima revisione di avanzamento (**`PB`**).

Queste cartelle sono organizzate nel modo seguente, (i termini in **`grassetto`** indicano il nome di una directory):

- **`1-Candidatura`**:
  - **`Verbali_Esterni`**;
  - **`Verbali_Interni`**;
  - `Dichiarazione_Degli_Impegni_V1.0.pdf`;
  - `Lettera_di_presentazione_V1.0.pdf`;
  - `Valutazione_Capitolati_V1.0.pdf`.
- **`2-RTB`**:
  - **`Documentazione_Esterna`**:
    - **`Verbali_Esterni`**;
    - `Analisi_dei_Requisiti_v1.0.0.pdf`;
    - `Piano_di_Progetto_v1.0.0.pdf`;
    - `Piano_di_Qualifica_v1.0.0.pdf`.
  - **`Documentazione_Interna`**:
    - **`Verbali_Interni`**;
    - `Glossario_v1.0.pdf`;
    - `Norme_di_Progetto_v1.0.0.pdf`.
- **`3-PB`**

3.2.4.3) Struttura della repository BuddyBot

In questa repository utilizziamo due branch principali:

- **`master`**: contiene l'ultima versione stabile del codice.
- **`develop`**: raccoglie le ultime modifiche in corso di sviluppo.

Le modifiche apportate in `develop` vengono generalmente integrate in `master` al termine di ogni periodo di lavoro.

Per il **`POC`** è stata creata una repository dedicata, **`BuddyBot---POC`**, che contiene il codice relativo al Proof of Concept.

3.2.4.4) Sito Vetrina

La repository SweeTenTeam.github.io ospita il codice del sito dedicato al progetto. Il sito ha l'obiettivo di fornire un'interfaccia veloce e intuitiva, nella quale poter visualizzare ergonomicamente e in maniera organizzata i documenti relativi al progetto stesso. Tra le funzionalità del sito, è inclusa anche la visualizzazione diretta del Glossario, che fornisce definizioni chiare e precise dei termini rilevanti. Questa sezione aiuta a prevenire fraintendimenti o interpretazioni ambigue e permette di accedere rapidamente alle definizioni cliccando sui termini presenti nei documenti.

3.2.5) Sincornizzazione

La sincronizzazione avviene attraverso *repository*$_G$ condivise su *GitHub*$_G$, dove ogni attività è tracciata tramite un'apposita *issue*$_G$. Per ogni documento o feature esiste un *branch*$_G$ dedicato, che è stato originariamente creato a partire dal *branch*$_G$ **`develop`**. Quando un membro del team inizia un nuovo task, crea un nuovo *branch*$_G$ a partire dal *branch*$_G$ specifico per il documento o la feature correlata al task. Tuttavia, per i *verbali* e il *glossario*, i *branch*$_G$ vengono creati direttamente a partire da **`develop`**, poiché questi documenti sono meno complessi e richiedono modifiche più leggere. Questo approccio consente di avere ambienti di lavoro indipendenti, permettendo di svolgere più attività in parallelo e ottimizzando la collaborazione all'interno del team.

3.2.5.1) Branch

La struttura dei branch descritta in precedenza si basa sulla metodologia single-purpose, che prevede la creazione di un *branch*$_G$ per ciascuna attività specifica. Questo approccio consente di suddividere il lavoro in compiti distinti, garantendo un flusso di lavoro stabile e organizzato. Una volta completata un'attività, il *branch*$_G$ corrispondente viene unito al *branch*$_G$ specifico per il documento o la feature correlata e successivamente eliminato. Una volta che il documento o la feature raggiunge uno stato coerente e stabile, il suo *branch*$_G$ specifico viene unito al «brach principale» (**`develop`**). È importante chiarire che **`master`** è il «branch di default», mentre per «branch principale» si intende il *branch*$_G$ in cui le modifiche devono essere integrate.

3.2.5.2) Pull request

Al termine di un'attività o di una sua parte, il membro incaricato apre una *Pull Request*$_G$, indicando il verificatore. Quest'ultimo, dopo aver convalidato le modifiche, esegue il *merge*$_G$ della *Pull Request*$_G$ nel *branch*$_G$ specifico per il documento o la feature correlata, chiude l'*issue*$_G$ associata ed elimina il branch di lavoro se non più necessario.

3.3) Verifica

3.3.1) Scopo ed Aspettative

La verifica nel ciclo di vita del *software*$_G$ è un'attività costante che parte dalla fase di progettazione e continua fino alla manutenzione. Questo processo è fondamentale per assicurare che gli output del *software*$_G$, come codice sorgente, documentazione e test, rispettino le specifiche e gli obiettivi definiti, basandosi su principi di coerenza, completezza e accuratezza.

Lo scopo primario è implementare un sistema di controllo per ogni prodotto, garantendo un'analisi efficace e risultati affidabili. Attraverso tecniche strutturate di analisi e test, si verifica la conformità dei prodotti ai requisiti prefissati. Procedure ben definite e criteri rigorosi, insieme alla validazione finale, rappresentano i pilastri per un processo di sviluppo corretto.

La solidità del prodotto, ottenuta grazie alla verifica, costituisce una base essenziale per facilitare la transizione alla fase di validazione e per garantire elevati standard di qualità.

3.3.2) Descrizione

La verifica è un'attività sistematica e continua, affidata ai verificatori, che si applica ai processi in corso per garantire il rispetto degli standard stabiliti. Questo processo si ripete ciclicamente e si adatta alle esigenze del progetto man mano che queste evolvono.

Il `Piano di Qualifica` rappresenta lo strumento centrale della verifica. Esso specifica gli obiettivi, i criteri di accettazione e i metodi necessari per eseguire un controllo accurato ed efficace.

La documentazione prodotta durante la verifica non è un semplice obbligo formale, ma uno strumento chiave per assicurare trasparenza, tracciabilità e ripetibilità. Nei punti seguenti verranno illustrate le attività da considerare per rendere la verifica più efficace.

3.3.3) Analisi statica

L'analisi statica è una metodologia di verifica che non richiede l'esecuzione del prodotto, ma si basa su una revisione approfondita del codice e della documentazione. Il suo scopo principale è garantire il rispetto dei requisiti, individuare eventuali errori e confermare che siano presenti tutte le caratteristiche richieste. Le due metodologie principali per condurre l'analisi statica, sono: l'*inspection*$_G$ e il *walkthrough*$_G$.

3.3.3.1) Inspection

Questa metodologia adotta un approccio strutturato per individuare potenziali difetti nel prodotto e nella documentazione. I difetti vengono inizialmente definiti e organizzati in specifiche liste di controllo, che vengono successivamente utilizzate in modo sistematico per valutare codice e documenti.

3.3.3.2) Walkthrough

A differenza del metodo *inspection*$_G$, il *walkthrough*$_G$ privilegia una verifica più collaborativa tra l'autore e il verificatore, ponendo maggiore enfasi su un confronto approfondito. Questo metodo non si limita a individuare problemi specifici, ma mira a garantire una correttezza complessiva, adattandosi meglio al contesto del documento o del codice analizzato. Sebbene richieda un impiego maggiore di risorse, il *walkthrough*$_G$ offre un livello più elevato di accuratezza. Per questo progetto, il team ha deciso di adottare questo approccio per la revisione della documentazione. Tale scelta nasce dall'esigenza di gestire un insieme di dati variegato e complesso, dove il metodo inspection potrebbe risultare troppo rigido per garantire una valutazione completa e adeguata.

3.3.4) Analisi dinamica

L'analisi dinamica si focalizza sul monitoraggio del comportamento del codice durante la sua esecuzione, per individuare eventuali errori. Questo tipo di metodologia è specifica per ogni progetto, poiché le attività di verifica, tra cui i test, sono determinati dal contenuto del codice e dai requisiti specifici.

I test eseguiti in questa fase permettono una valutazione oggettiva del codice e delle sue funzionalità, poiché, essendo ripetibili, forniscono gli stessi risultati ogni volta, senza influenze da fattori esterni. La progettazione e l'esecuzione dei test si basano sui principi del *Modello a V*$_G$.

3.3.4.1) Test di unità

I *Test di unità*$_G$ si concentrano sulle singole componenti autonome del *software*$_G$ e vengono generalmente implementati durante la fase di progettazione.

Questi test si suddividono in due categorie, a seconda del tipo di verifica da effettuare:

- **Test funzionali**: accertano che l'output generato corrisponda ai risultati attesi.
- **Test strutturali**: esaminano i vari percorsi del codice, verificando la copertura di tutti i cammini possibili attraverso una serie di test mirati.

3.3.4.2) Test di integrazione

I *Test di integrazione*$_G$ vengono definiti durante la fase di progettazione dell'architettura, successivamente ai *Test di unità*$_G$. Questi test, con un approccio graduale, verificano la corretta integrazione tra le diverse unità *software*$_G$ precedentemente testate. In caso di errori, è possibile annullare le modifiche apportate, ripristinando così uno stato sicuro.

Esistono due principali approcci per condurre i *Test di integrazione*$_G$:

- `Top-down`: si inizia testando le componenti di sistema con maggiori dipendenze e visibilità esterna, così da rendere subito disponibili le funzionalità di alto livello. Questo approccio favorisce una verifica approfondita delle funzionalità centrali.
- `Bottom-up`: si parte dalle componenti con minori dipendenze e maggiore rilevanza interna, quelle meno visibili all'utente, per testare prima i componenti più basilari e meno complessi.

3.3.4.3) Test di sistema

I *Test di sistema*$_G$ vengono progettati dopo i *Test di integrazione*$_G$. Questi test sono finalizzati a garantire il corretto funzionamento complessivo del sistema, verificando che tutti i requisiti *software*$_G$, definiti nel capitolato e redatti nel documento *Analisi dei Requisiti*$_G$, siano implementati e operativi.

3.3.4.4) Controlli di regressione

I controlli di regressione verificano che le modifiche apportate a specifiche componenti architetturali, come correzioni o espansioni, non compromettano il funzionamento dell'intero sistema.

Questi controlli consistono nella ripetizione mirata di:

- *Test di unità*$_G$;
- *Test di integrazione*$_G$;
- *Test di sistema*$_G$.

Questi sono fondamentali per assicurare che le funzionalità precedentemente verificate rimangano intatte e non si verifichino regressioni nel comportamento del sistema.

3.3.4.5) Test di accettazione

I *Test di accettazione*$_G$ hanno lo scopo di garantire che il prodotto finale soddisfi le richieste e le aspettative stabilite dalla *Proponente*$_G$. Per questa ragione, devono essere svolti obbligatoriamente in sua presenza.

3.3.4.6) Identificazione e stato dei test

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

`T[Tipologia]-[Numero]`

Dove **`Tipologia`** indica il tipo di test:

- **`U`**: Test di unità;
- **`I`**: Test di integrazione;
- **`S`**: Test di sistema;
- **`A`**: Test di accettazione.

Ogni test ha anche uno **`stato`** che ne indica l'esito:

- **`V`**: Verificato (test superato);
- **`NV`**: Non Verificato (test fallito);
- **`NI`**: Non Implementato (test non eseguito).

3.4) Validazione

3.4.1) Scopo ed aspettative

La validazione rappresenta una fase cruciale nello sviluppo del *software*$_G$, che si assicura che il prodotto finale sia in linea con i requisiti e le esigenze del cliente.

Questo processo si concentra su quattro aspetti principali:

- **Conformità ai requisiti**: il prodotto deve rispettare tutte le specifiche definite;
- **Correttezza operativa**: il prodotto deve operare correttamente, in conformità con la logica di progettazione;
- **Usabilità**: il prodotto deve essere intuitivo e semplice da utilizzare per gli utenti;
- **Efficacia**: il prodotto deve dimostrarsi efficace nel soddisfare le necessità dei clienti.

L'aspettativa finale è quella di ottenere un prodotto che soddisfi completamente i requisiti definiti e funzioni in modo corretto.

3.4.2) Descrizione

Durante la fase di validazione, faremo riferimento ai test precedentemente eseguiti durante la verifica, come indicato nelle relative sezioni delle **`Norme di Progetto`**. Il processo di validazione del prodotto si concluderà con l'esecuzione dei test di accettazione, che ne confermeranno la conformità.

3.5) Gestione della qualità

3.5.1) Scopo

Il processo di Gestione Della Qualità ha come obiettivo principale quello di garantire che i processi, i prodotti e gli artefatti tutti nel ciclo di vita del progetto, siano conformi ai piani specificati, allineati con i requisiti stabiliti e che, in generale, rispettino i canoni di qualità in precedenza decisi.

3.5.2) Descrizione

Per garantire e mantenere un certo standard di qualità, il team si impegna ad utilizzare altri processi di supporto, come ad esempio la *Verifica*$_G$, *Validazione*$_G$ e revisioni aggiuntive con il *Proponente*$_G$. Una volta stabiliti e definiti gli standard da rispettare nel *Piano Di Qualifica*$_G$, rimane solamente da assicurarsi che questi vengano applicati e rispettati in ogni fase del progetto. Sono state quindi definite le `best practices` di modo che ogni componente del gruppo lavori seguendo le stesse direttive e linee guida. In questo modo, applicare e rispettare questi standard diventa più semplice e veloce. Per

controllare che questi standard siano applicati durante tutta la durata del progetto, ogni prodotto o artefatto è controllato da un *Verificatore*$_G$.

3.5.3) Piano Di Qualifica

Il documento **Piano di Qualifica** è fondamentale per garantire il rispetto gli standard di qualità richiesti, per il completamento degli obiettivi di questo *processo*$_G$ e per soddisfare le richieste e le aspettative degli *Stakeholders*$_G$.

Il documento si estende, e non si limita, a:

- Definire gli obiettivi di qualità del progetto;
- Definire le metriche di visione quantitativa;
- Definire test di qualità e di accettazione;
- Fornire una retrospettiva sul progetto;

3.5.4) Testing

Il suddetto documento **Piano di Qualifica** definisce obiettivi di qualità sia dei processi che dei prodotti. Le metriche relative forniscono una verifica sugli aspetti di accessibilità, mentre i test garantiscono la qualità generale del software e dei processi.

Le categorie di test sono:

1. *Test di unità*$_G$: verifica il corretto sviluppo e funzionamento di un'unica unità del sistema. Un'unità è definita come un elemento indivisibile e unico del sistema stesso.
2. *Test di integrazione*$_G$: verifica il corretto funzionamento di più unità del sistema, integrate tra loro, ma che cooperano per svolgere un'unica funzione all'interno del sistema.
3. *Test di sistema*$_G$: verifica il funzionamento del sistema nel suo intero. I requisiti funzionali, di vincolo, di qualità e di prestazione precedentemente concordati con il Proponente sono testati e verificati in questa fase.
4. *Test di accettazione*$_G$: verifica il soddisfacimento del Proponente rispetto ai requisiti concordati. Questi test sono eseguiti in presenza del Proponente stesso. Il superamento di questi test garantisce il rilascio del prodotto.

3.5.5) Metriche

Le metriche di qualità sono fondamentali per garantire il rispetto degli standard di qualità richiesti. Il documento **Piano di Qualifica** fornisce varie metriche utilizzate per misurare e rispettare gli standard di qualità.

Le metriche sono divise in due categorie:

- Processo: PRC;
- Prodotto: PRD.

Inoltre, le suddette metriche sono identificate come segue:

`M-[Categoria-Metrica]-[Sigla-Identificativa-Metrica]`

Ogni metrica ha:

- un valore minimo (o accettabile): valore sotto il quale il processo o il prodotto misurato diventa inaccettabile;
- un valore ammissibile (o ottimale): valore ideale che dovrebbe essere raggiunto dalla metrica;
- descrizione: descrizione breve della metrica.

3.5.6) Aspettative

Il team si aspetta di rispettare in ogni momento gli standard di qualità definiti. Inoltre, il team si aspetta:

- Ottima qualità del prodotto;
- Ottima qualità dei processi;
- Miglioramento costante;
- Test frequenti e predicibili;
- Soddisfazione delle richieste e aspettative del Proponente.

4) Processi Organizzativi

I processi organizzativi sono fondamentali per garantire il corretto svolgimento del progetto in conformità con gli standard di qualità prefissati.

4.1) Gestione dei Processi

4.1.1) Scopo

Come stabilito dallo Standard `ISO/IEC 12207:1997`, il processo organizzativo di gestione dei processi ha come scopo quello di identificare le attività generali che ogni membro del team è chiamato a svolgere. È un'attività chiave per garantire il completamento del progetto in modo efficiente e in linea con gli standard di qualità già definiti e con le aspettative del Proponente.

4.1.2) Descrizione

Il processo è diviso nelle seguenti attività:

- Inizio e definizione dello Scopo;
- Pianificazione;
- Esecuzione e Controllo;
- Revisione e Valutazione;
- Chiusura.

4.1.3) Pianificazione

4.1.3.1) Scopo

Come da Standard `ISO/IEC 12207:1997`, il ***Responsabile***$_G$ ha il compito di predisporre i piani per l'esecuzione di tutte le attività di pianificazione. I piani dovranno essere dettagliati, contenere la descrizione delle attività e dei compiti (e ruoli) assegnati. Il Responsabile dovrà redigere questa pianificazione all'interno del documento ***Piano di Progetto***$_G$. Questo documento riporta tutte le descrizioni e pianificazioni delle attività da svolgere in un certo periodo (uno o più sprint).

4.1.3.2) Descrizione

L'attività di pianificazione viene articolata nelle due seguenti sezioni:

1. Ruoli;
2. Metodo di Lavoro;
3. Ticketing.

4.1.3.3) Aspettative

L'attività di pianificazione aiuta il team a:

- Organizzare e pianificare le attività da svolgere;
- Assegnare i compiti ai membri del team;
- Rispettare le scadenze prefissate;
- Monitorare e controllare l'avanzamento del progetto.

Inoltre serve al Team per rispettare le regole organizzative stabilite per lo svoglimento del progetto.

4.1.3.4) Ruoli

I ruoli sono decisi dal Responsabile di Progetto. Questa figura è coordinatrice di tutti i membri, e delle attività da essi svolte. Al termine del progetto, ogni memebro dovrà aver ricoperto ogni ruolo. **SweeTenTeam** ha deciso di ruotare i ruoli ogni sprint (ogni due settimane).

I ruoli in depth:

Responsabile di progetto
Ha il compito fondamentale di rappresentare il gruppo e di esporlo al Proponente e ai committenti.

I compiti di questo ruolo sono:

- Approvare la documentazione;
- Gestire la pianificazione del progetto;
- Coordinare i membri del gruppo assegnando tasks e ruoli;
- Studiare e gestire l'analisi dei rischi;
- Coordinare e curare i rapporti esterni ed interni.

Amministratore
Definisce, controlla e gestisce l'ambiente e gli strumenti di lavoro del progetto. Ha piena responsabilità sull'efficacia ed efficienza del ***Way of Working***$_G$.

I pricipali compiti sono:

- Redigere e mantenere aggiornata la documentazione: gestisce il versionamento;
- Gestire la configurazione del progetto e del prodotto: controllo sul prodotto software;
- Gestire i processi: problem solving rispetto ai processi;
- Migliorare l'ambiente di lavoro: ricerca gli strumenti e le tecnologie necessarie per migliorare e automatizzare il lavoro.

Analista
L'***Analista***$_G$ approfondisce le richieste, tecniche e non, del ***Capitolato***$_G$. E' principalmente fondamentale, e presente, nella prima fase del progetto, quando viene redatta l'***Analisi dei Requisiti***$_G$. Supervisiona la scrittura della stessa, in quanto è di vitale importanza che non vi siano errori. Avere errori nell'***Analisi dei Requisiti***$_G$ comprometterebbe l'intero progetto.

Ha il compito di:

- Studiare il progetto e il suo contesto applicativo;
- Raccogliere, analizzare e studiare i bisogni del cliente;
- Scrivere l'***Analisi dei Requisiti***$_G$
- Definire la complessità dei vari requisiti.

Progettista
Determina le scelte tecniche del progetto, trasformando i requisiti trovati dagli Analisti in un'architettura che strutturi e rappresenti il problema. Il progettista inoltre seguirà lo sviluppo ma non la manutenzione.

I compiti principali sono:

- Sviluppare un prodotto economico e facilmente manutenibile partendo dalle indicazioni degli Analisti;
- Favorire efficienza ed efficacia con le sue scelte.

Verificatore
Controlla il lavoro svolto dagli altri membri del Team e si assicura che sia conforme agli standard di qualità definiti.

Il Verificatore ha il compito di:

- Verificare che le attività svolte siano corrette e conformi agli standard attraverso le tecniche e gli strumenti definiti nelle `Norme di Progetto`.

Programmatore
Svolge l'attività di codifica del progetto e delle sue componenti di supporto basandosi sull'architettura indicata dal Progettista.

I compiti principali sono:

- Implementare le funzionalità richieste dal Progettista;
- Scrivere codice pulito, documentato e manutenibile che rispetti le `Norme di Progetto`;
- Creare Test per verifica e validazione del codice;
- Scrivere il ***manuale utente***$_G$.

4.1.3.5) Metodo di Lavoro

Per lo svoglimento del progetto, SweeTenTeam ha deciso di adottare il modello di sviluppo ***Agile***$_G$, in particolare il framework ***Scrum***$_G$. Questo modello prevede la suddivisione del progetto in periodi di tempo definiti, chiamati ***Sprint***$_G$. Gli sprint garantiscono una corretta pianificazione delle attività da svolgere e una maggiore modularità di esse, in quanto permettono una maggiore suddivisione in «sotto-attività».

Ogni Sprint è caratterizzato dalle seguenti fasi:

- **Sprint Planning**: definizione degli obiettivi e delle attività da svolgere assegnandole ad ogni membro. Queste attività sono scelte in base all'incontro SAL precedentemente svolto con il Proponente. Inoltre, vengono discusse le attività rimaste dallo sprint precedente. Vengono poi create le *Issues* su GitHub e assegnate ai membri del Team.
- **Svolgimento**: nell'arco delle due settimane le issue vengono portate a termine dai membri del team assegnati. Si veda il paragrafo del workflow per maggiori dettagli.
- **Revisione**: Alla fine dello sprint, viene effettuata la revisione delle varie attività completate e quelle che sono rimaste attive. Questo permette di avere una visione chiara dell'avanzamento del progetto e di eventuali problemi riscontrati.
- **Retrospettiva**: viene valutato l'andamento generale dello sprint oltre che all'effettivo lavoro svolto. Vengono discusse le difficoltà riscontrate e le soluzioni adottate.

4.1.3.6) Ticketing

Il Team utilizza il sistema di ***Issue Tracking Sistem***$_G$ (ITS) di GitHub. Questo permette di assegnare compiti e ruoli ai membri del Team, di monitorare l'avanzamento del progetto e di tenere traccia delle attività svolte, oltre ad essere facilmente collegabile a ***Pull Request***$_G$ e ***Commit***$_G$.

Le issue principali sono create dal responsabile del gruppo che assegna i compiti ai vari membri del team. In seguito, ogni problema è ulteriormente suddiviso in sotto-problemi e di conseguenza, «sotto-issue». Queste ultime sono create non dal Responsabile, ma dall'assegnatario della issue principale. Questo permette di avere una visione chiara e dettagliata delle attività da svolgere e di monitorare l'avanzamento del progetto. Ad esempio:

- Responsabile crea la Issue `Norme di Progetto - Processi Primari` e la assegna al componente X.

- X studia il documento `Norme di progetto` e, se lo ritiene opportuno, individua sotto-issue per rendere il problema più approcciabile.
- X crea le sotto-issue individutae, le auto-asssegna e sceglie il verificatore.

Ogni *Issue* è composta da:

- Titolo: titolo della issue;
- Descrizione: descrizione dell'attività da completare;
- Assegnatario (o Assegnatari): chi deve svolgere questa issue;
- Verificatore: chi deve verificare il lavoro svoltop;
- ***Milestone***$_G$: traguardo da raggiungere;
- ***Label***$_G$: etichetta per categorizzare le issue;
- ***Project***$_G$: progetto a cui appartiene la issue;
- stato: aperta, chiusa, in corso, ecc.
- data di inizio: data di inizio dell'attività;
- data di fine: data di fine dell'attività.

La creazione e la gestione della issue segue il seguente flusso:

1. Creazione della Issue;
2. Assegnazione della Issue;
3. Creazione delle sotto-issue;
4. Assegnazione delle sotto-issue;
5. Inizio dell'attività;
6. Spostamento della Issue da `To Do` a `In Progress`;
7. Svolgimento attività;
8. Fine attività;
9. Creazione ***Pull Request***$_G$
10. Verifica del lavoro svolto:
    11. se corretto, si passa al punto 13;
    12. se non corretto, il verificatore lascia commenti sulla ***Pull Request***$_G$ riguardanti le modifiche necessarie, l'assegnatario della issue apporta le modifiche e si ripete dal punto 10.
13. Chiusura della Issue;
14. Spostamento della Issue da `In Progress` a `Done`;
15. Chiusura ***Pull Request***$_G$ e merge sul branch specifico per il documento o la feature correlata alla Issue;
16. Eventuale eliminazione del branch (solo se non deve essere riutilizzato).

Milestones

Il gruppo, dopo una pianificazione accurata, ha deciso di impostare le milestones ad "***Artefatti***$_G$" e non a "***sprint***$_G$". Il gruppo ritiene che le Milestones indichino qualcosa di completato, quindi è stato deciso di ritenere una milestone completata solamente quando un artefatto è stato completato e verificato. Al contrario, è stata impostata una data di inizio e una data di fine alla singola issue, in modo che l'avanzamento del progetto sia comunque monitorabile nel tempo.

4.1.4) Coordinamento

È l'attività responsabile della gestione delle comunicazioni e degli incontri, sia interni che esterni. Questo compito assume un ruolo fondamentale nella corretta riuscita del progetto, garantendo l'efficienza e l'efficacia del Team stesso.

Le attività di coordinamento comprendono le comunicazioni interne ed esterne, l'organizzazioni delle riunioni e la gestione delle scadenze.

4.1.4.1) Comunicazioni

4.1.4.1.1) Interne

Le comunicazioni interne avvengono con due modalità: le comunicazioni scritte, formali e non, avvengono attraverso *Telegram*$_G$. Le comunicazioni verbali, quindi le riunioni, la condivisione di file, di codice, e di qualsiasi altra risorsa avvengono attraverso *Discord*$_G$.

Su *Telegram*$_G$ sono affrontate solamente questioni di rapida risoluzione, e comunicazioni off-topic. Le decisioni più strutturate, come elencato sopra, verranno prese su discord in presenza di tutti i memebri del Team durante la riunione settimanale.

4.1.4.1.2) Esterne

Tutte le comunicazioni esterne sono gestite dal Responsabile del Progetto, previa consultazione con il Team e visione della comunicazione stessa. Le comunicazioni formali vengono effettuale tramite email attraverso questo indirizzo: sweetenteam@gmail.com. L'azienda azzurrodigitale: ha messo a disposizione anche un canale di comunicazione per facilitare il dialogo con i responsabili del progetto assegnato al team.

4.1.4.2) Riunioni

Il responsabile attuale ha il compito di introdurre i punti che saranno trattati nella riunione e di redigere il verbale interno, oltre che a decidere chi lo verificherà. La riunione avviene su Discord, in un canale dedicato.

4.1.4.2.1) Interne

Le riunioni interne avvengono settimanalmente, il giorno e l'ora sono decisi dal Responsabile e comunicati al Team in base alle esigenze di ogni membro. Queste riunioni servono per discutere lo stato del progetto, le attività svolte e da svolgere, e per prendere decisioni importanti.

4.1.4.2.2) Esterne

Le riunioni esterne che coinvolgono il Proponente e i Committenti si tengono generalmente ogni due settimane, il martedì dalle 17:00 alle 18:00 (con eventuali modifiche di data o orario in caso di necessità). Le riunioni esterne sono effettuate su *Google Meet*$_G$. Queste riunioni sono gestite dal Responsabile del Progetto, che si occupa di organizzare l'incontro, di redigere l'ordine del giorno e il verbale esterno. Quest'ultimo viene condiviso con il Team e con il Proponente.

4.1.4.3) Verbali

Dopo ogni incontro, sia esso interno od esterno, viene redatto un verbale dal Responsabile e verificato dal Verificatore stabilito.

4.1.4.3.1) Interni

In ogni incontro interno vengono affrontati diversi temi, come si vede al punto 4.1.4.2.1. Al termine di una riunione, viene creata dal responsabile una issue, che è auto-assegnata. Dopo la redazione di tale verbale, viene verificato, creata la *Pull Request*$_G$ e chiusa la issue.

4.1.4.3.2) Esterni

Le modalità sono le stesse del punto precedente. Dopo che la issue viene chiusa, il verbale viene mandato al Proponente e ai Committenti per essere firmato e approvato. Dopo la firma, il verbale viene caricato su *GitHub*$_G$.

4.2) Infrastruttura

Fanno parte dell'Infrastruttura organizzativa tutti gli strumenti che il Team utilizza per la gestione del progetto. Questi strumenti sono fondamentali per garantire la corretta riuscita del progetto e per mantenere un alto livello di efficacia ed efficienza. Tali strumenti permettono la comunicazione, il coordinamento e la pianificazione.

4.2.1) Strumenti

4.2.1.1) GitHub

È la piattaforma principale utilizzata dal Team per la gestione e il controllo di versione del progetto. Il *Workflow*$_G$ utilizzato dal Team è GitHub Flow, che è un modello di sviluppo leggero che prevede l'utilizzo di branch per lo sviluppo di nuove funzionalità e la creazione di *Pull Request*$_G$ per la loro integrazione.

In particolare, il flow si espande come segue:

- Riallineamento della *repository*$_G$ locale con quella remota;
- Creazione di un *branch*$_G$ locale per lo sviluppo di una singola feature, partedo dal branch specifico per il documento o la feature correlata al task (ogni branch segue la nomenclatura `[tipo-feature]/[nome-feature]-[numero_issue]`, ad esempio `doc/norme-di-progetto-32`);
- Per i verbali e il glossario, il *branch*$_G$ viene creato direttamente a partire da develop, poiché questi documenti richiedono modifiche più leggere (ogni branch dei verbali segue la nomenclatura `doc/[verbale]/[tipo-verbale]-[data]`, ad esempio `doc/verbale/interno-11-28`);
- Push del branch locale sulla repository remota;
- Creazione di una *Pull Request*$_G$ per la feature;
- Verifica della *Pull Request*$_G$ da parte del/dei Verificatore/i;
- Merge ed eliminazione del *branch*$_G$.

Viene anche utilizzata la *project board*$_G$ di GitHub per la gestione delle issue e delle milestone.

La board possiede le seguenti viste:
- *KanBan*:
  - To Do: issue appena create e da completare;
  - In Progress: issue in corso di svolgimento;
  - Verify: issue completate e in attesa di verifica;
  - Done: issue completate e verificate.

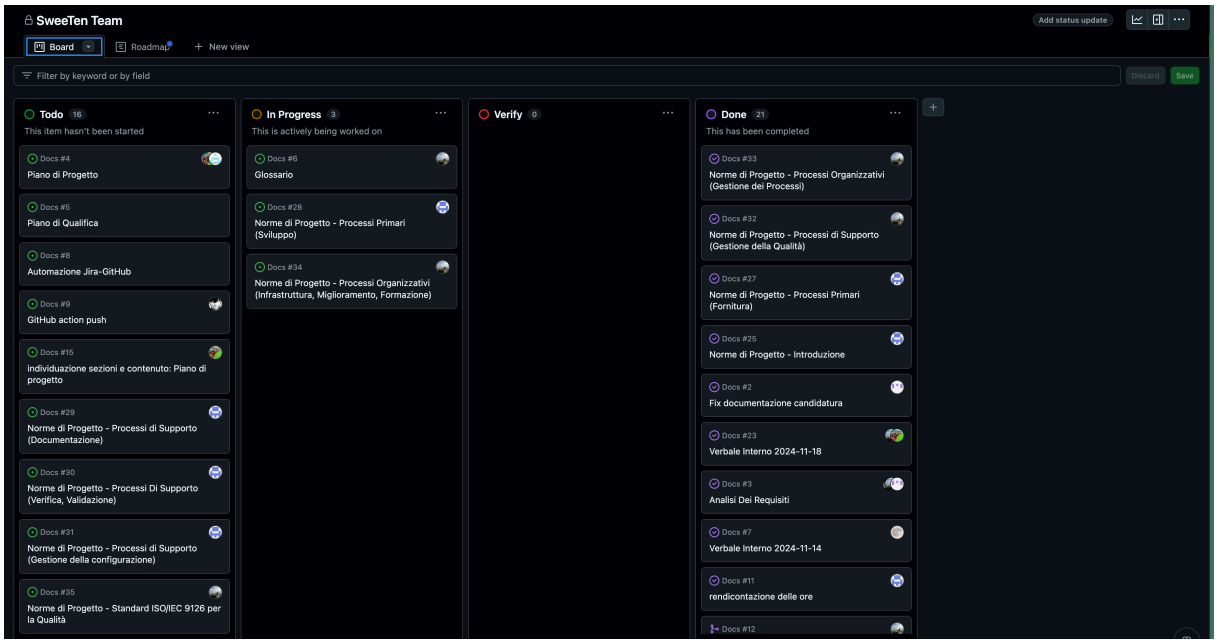

Figura 16: Kanban Board aggiornata al 22/11/2024

- *Roadmap*:
  - Start Date: data di inizio della issue;
  - End Date: data di fine della issue.

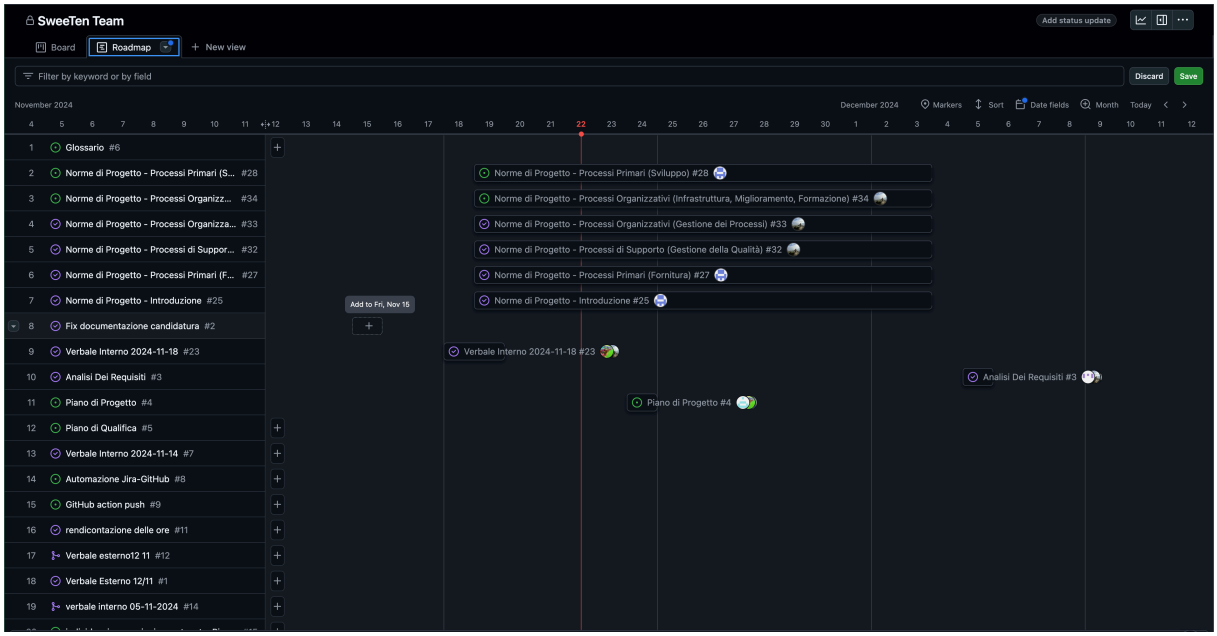

Figura 17: Roadmap aggiornata al 22/11/2024

Per quanto concerne le *Issue* si veda il capitolo **4.1.3.5) Ticketing**. Le *Issue* non vengono mai eliminate, ma solamente chiuse (GitHub le sposta automaticamente nella sezione `Closed`).

4.2.1.2) Telegram

È il principale strumento di comunicazione interna scritta utilizzato dal Team. Viene utilizzato per comunicazioni rapide e informali, per condividere link e per discutere di questioni di minore importanza.

4.2.1.3) Discord

È la piattaforma utilizzata dal Team per le comunicazioni sincrone, che siano verbali o scritte e per la condivisione di file, dati, link e appunti.

In un canale apposito, viene utilizzato anche per comunicazioni informali o urgenti con il proponente.

4.2.1.4) Google Meet

È la piattaforma utilizzata per le riunioni esterne sincrone con il Proponente e i Committenti.

4.2.1.5) Google Calendar

È il servizio utilizzato per la gestione delle riunioni esterne con il Proponente e i Committenti. Viene utilizzato per fissare le date delle riunioni e per inviare gli inviti ai partecipanti.

4.2.1.6) Google Drive

È il servizio utilizzato per la condivisione di documenti e file tra i membri del Team. In particolare, vengono condivisi i fogli per la rendicontazione delle ore di lavoro e i termini da inserire nel glossario.

4.2.1.7) Gmail

È il servizio di posta elettronica utilizzato per le comunicazioni formali con il Proponente e i Committenti.

4.3) Miglioramento

4.3.1) Scopo

Rappresenta un procedimento atto ad istituire, valutare, controllare e ottimizzare il ciclo di vita del software. Si applica un processo ciclico, dove i ruoli dei membri cambiano ogni sprint, e dove le fasi vengono costantemente migliorate.

4.3.2) Descrizione

Il processo di miglioramento è costituito da tre fasi:

- Stabilimento dei processi;
- Valutazione dei processi;
- Miglioramento dei processi.

4.3.2.1) Stabilimento dei processi

Inizialmente, si devono stabilire dei processi organizzativi atti a migliorare efficacia ed efficienza del prodotto. Questi processi devono essere definiti in modo chiaro e preciso, in modo che ogni membro del Team possa seguirli senza problemi e documentarli in modo che possano essere facilmente consultabili.

4.3.2.2) Valutazione dei processi

Come anticipato, sviluppare, documentare e applicare una procedura di valutazione del processo. Questa viene eseguita basandosi sugli obiettivi del processo stesso, sulle metriche di qualità e sulle aspettative del Proponente.

4.3.2.3) Miglioramento dei processi

Infine, individuati i processi da migliorare, si applicano le modifiche necessarie per ottimizzare il processo stesso. Di conseguenza, si deve aggiornare la documentazione relativa al processo di modo che rifletta le modifiche apportate.

4.3.3) Metriche

La misurazione del miglioramento interessa diverse aree, caratterizzate a loro volta da diverse metriche che riflettono aspetti chiave del processo stesso.

Individuiamo le seguenti:

- Velocità di sviluppo;
- Tasso di errori;
- Conformità agli standard.

4.3.4) Strumenti

Per monitorare i miglioramenti, il gruppo utilizza il seguente strumento:

- ***GitHub***$_G$: per la gestione del codice e delle issue, e per la creazione di ***Pull Request***$_G$ e ***Branch***$_G$. GitHub permette un tracciamento sia temporale che qualitativo del lavoro svolto.

4.4) Formazione

4.4.1) Scopo

Il processo di formazione ha come obiettivo quello di garantire che ogni membro del Team abbia le competenze necessarie per svolgere il proprio ruolo in modo efficace ed efficiente. Questo processo è fondamentale per garantire il corretto svolgimento del progetto e per il suo successo.

4.4.2) Descrizione

Al fine di promuovere un ambiente di lavoro efficace ed efficiente, che garantisce un processo organico, ogni membro del Team si impegna autonomamente a:

- Individurare le proprie lacune riguardanti le tecnologie e i requisiti del progetto;
- Colmare queste lacune attraverso lo studio autonomo e asincrono;
- Condividere le proprie conoscenze con gli altri membri del Team.

4.4.3) Aspettative

È previsto che ogni membro del Team acquisisca le competenze e conoscenze necessarie al corretto e efficace svolgimento del progetto, oltre che le competenze trasversali necessarie per il lavoro di gruppo.

4.4.4) Strumenti

Al fine di completare questo percorso formativo, ogni membro utilizzerà tutti gli strumenti che ritiene necessari e li condividerà tramite link o file con gli altri membri del Team attraverso i canali di comunicazione interna dedicati al fine di promuovere la condivisione delle conoscenze.

5) Standard ISO/IEC 9126

Lo standard ISO/IEC 9126 è una norma internazionale che ha contribuito a definire i parametri fondamentali per la valutazione della qualità e la validazione del software. Questo standard rappresenta un insieme di norme e linee guida dettagliate, oltre che a criteri di valutazione, per gli attributi di qualità di un prodotto software. Andando più nel dettaglio, questo standard individua sei categorie di attributi di qualità. Ognuna di queste categorie, è a sua volta suddivisa in sotto-categorie, che rappresentano i parametri di ***valutazione*** e ***misurazione*** per la qualità del software.

Queste categorie sono:

- ***Funzionalità***;
- ***Affidabilità***;
- ***Usabilità***;
- ***Efficienza***;
- ***Manutenibilità***;
- ***Portabilità***.

5.1) Funzionalità

Questo parametro si concentra sulla valutazione della capacità del software di fornire le funzionalità e le richieste specificate dal proponente.

Le sotto-categorie di questo parametro sono:

- ***Adeguatezza***: Capacità del software di fornire funzionalità che soddisfano i requisiti specificati e il contesto d'uso.
- ***Accuratezza***: Capacità del software di fornire risultati corretti o concordati con i requisiti specificati e il grado di precisione richiesto.
- ***Interoperabilità***: Capacità del software di interagire efficientemente ed efficacemente con uno o più sistemi specificati.
- ***Conformità***: Capacità del software di aderire a standard, convenzioni e regolamentazioni specificate.
- ***Sicurezza***: Capacità del software di proteggere informazioni e dati da accessi non autorizzati e da malfunzionamenti.

5.2) Affidabilità

Questo parametro si concentra sulla valutazione della capacità del software di mantenere un certo livello di prestazioni nel tempo anche in presenza di malfunzionamenti.

Le sotto-categorie di questo parametro sono:

- ***Maturità***: Capacità del software di evitare malfunzionamenti, guasti o errori in fase di esecuzione.
- ***Tolleranza agli errori***: Capacità del software di mantenere un certo livello di prestazioni anche in presenza di malfunzionamenti o errori.
- ***Recuperabilità***: Capacità del software di ripristinare un certo livello di prestazioni e di recuperare i dati in caso di malfunzionamenti.
- ***Affidabilità delle informazioni***: Precisione e affidabilità delle informazioni ritornate dal software.

5.3) Usabilità

Questo è un aspetto fondamentale della qualità del software in quanto influisce direttamente sulla user experience durante l'interazione con il prodotto.

Le sotto-categorie di questo parametro sono:

- ***Comprensibilità***: Capacità del software di essere compreso facilmente dall'utente.
- ***Apprendibilità***: Capacità del software di essere appreso facilmente dall'utente al primo utilizzo.
- ***Operabilità***: Facilità con cui un utente può portare a termine le operazioni desiderate in modo efficiente, senza incorrere in problemi e con un utilizzo normale di risorse.
- ***Attrattività***: Capacità dell'interfaccia e del design del software di essere piacevole e attraente per l'utente.
- ***Aderenza all'usabilità***: Capacità del software di aderire e rispettare nel suo ciclo di vita standard e convenzioni di usabilità.

5.4) Efficienza

Si concentra sulla capacità del prodotto di avere un utilizzo delle risorse ottimale ed efficiente, oltre che di garantire prestazioni elevate e tempi rapidi di risposta.

Le sotto-categorie di questo parametro sono:

- ***Comportamento rispetto al tempo***: Capacità del software di fornire risposte in tempi accettabili.
- ***Utilizzo delle risorse***: Capacità del software di utilizzare in modo efficiente le risorse a disposizione.
- ***Aderenza all'efficienza***: Capacità del software di aderire e rispettare nel suo ciclo di vita standard e convenzioni di efficienza.

5.5) Manutenibilità

Questo parametro si concentra sulla capacità del software di essere modificato, corretto e migliorato nel tempo.

Le sotto-categorie di questo parametro sono:

- ***Analizzabilità***: Capacità del software di essere analizzato facilmente per individuare errori e problemi o sezioni da migliorare.
- ***Modificabilità***: Capacità del software di essere modificato facilmente per correggere errori, migliorare le prestazioni e adattarsi a nuove esigenze senza generare nuovi errori.
- ***Stabilità***: Capacità del software di mantenere un certo livello di prestazioni anche in presenza di modifiche, evitando di compromettere il codice esistente.
- ***Testabilità***: Capacità del software di essere testato facilmente per garantire la qualità e la correttezza del prodotto e delle modifiche apportate.
- ***Aderenza alla manutenibilità***: Capacità del software di aderire e rispettare nel suo ciclo di vita standard.

5.6) Portabilità

Questo parametro si concentra sulla capacità del software di essere trasferito da un ambiente di sviluppo a un altro senza dover subire modifiche.

Le sotto-categorie di questo parametro sono:

- ***Adattabilità***: Capacità del software di essere adattato facilmente a diversi ambienti di sviluppo.
- ***Installabilità***: Capacità del software di essere installato facilmente in diversi ambienti.
- ***Sostituibilità***: Capacità del software di essere sostituito facilmente da un altro software nello stesso ambiente.

- ***Coesistenza***: Capacità del software di coesistere all'interno di un ambiente o sistema condiviso con altri applicativi.
- ***Conformità alla portabilità***: Capacità del software di aderire e rispettare nel suo ciclo di vita standard e convenzioni di portabilità.

6) Metriche Di Qualità Del Processo

Le metriche seguono la struttura definita a questo paragrafo. In questo capitolo, la prima parte della struttura sarà:

M-PRC-[S.I.]

6.1) Processi Primari

6.1.1) Fornitura:

- **M-PRC-EV** - Earned Value: valore del lavoro effettivamente completato rispetto al piano.
- **M-PRC-PV** - Planned Value: valore pianificato del lavoro in un determinato periodo di tempo.
- **M-PRC-AC** - Actual Cost: costo effettivamente sostenuto per completare il lavoro in un determinato periodo temporale.
- **M-PRC-CV** - Cost Variance: differenza tra il EV ed AC. Se è negativo, si è sforato il budget.
- **M-PRC-SV** - Schedule Variance: indica la differenza tra il EV e PV.
- **M-PRC-EAC** - Estimated at Completion: stima del costo totale basata sul rendimento attuale.

$$\mathrm{EAC} = \mathrm{ETC} + \mathrm{AC}$$

- **M-PRC-ETC** - Estimate to Complete: stima dei costi rimanenti.

$$\mathrm{ETC} = \mathrm{BAC} - \mathrm{EV}$$

- **M-PRC-BAC** - ***Budget at Completion***$_G$.

6.1.2) Sviluppo

- **M-PRC-RSI** - Requirements Stability Index: stabilità dei requisiti nel tempo.

$$\mathrm{RSI} = 100 - \left(\frac{\mathrm{RM} + \mathrm{RA} + \mathrm{RR}}{\mathrm{NR}}\right) * 100$$

  - **RA**: numero di requisiti aggiunti nel periodo considerato;
  - **RM**: Requisiti modificati;
  - **RR**: Requisiti rimossi;
  - **NR**: Numero di requisiti al momento dell'analisi.

- **M-PRC-SFI** - Structural Fan In: quantità di componenti che sfruttano un modulo specifico.
- **M-PRC-SFO** - Structural Fan-Out: componenti utilizzate dal modulo in osservazione.

6.2) Processi di supporto

6.2.1) Documentazione

- **M-PRC-GLP** - Gulpease Index: Valuta la leggibilità.

$$\mathrm{GLP} = 89 + \frac{300 * N_f - 10 * N_l}{N_p}$$

  - $\mathbf{N_f}$: Numero frasi;
  - $\mathbf{N_p}$: Numero parole;
  - $\mathbf{N_l}$: Numero lettere.

- **M-PRC-CO** - Correttezza Ortografica.

6.2.2) Verifica

- **M-PRC-CC** - Code Coverage: Percentuale di codice testato rispetto al totale.

6.2.3) Gestione Qualità

- **M-PRC-QMS** - Quality Metrics Satisfied: grado di soddisfacimento delle metriche di qualità.

6.3) Processi Organizzativi

- **M-PRC-NCR** - Non-Calculated Risk: monitoraggio dei rischi non inclusi nelle stime.
- **M-PRC-TE**: - Temporal Efficiency: Monitoraggio dell'efficienza temporale.

$$\mathrm{TE} = \frac{O_{\text{orologio}}}{O_{\text{produttive}}}$$

7) Metriche Di Qualità Del Prodotto

Come al paragrafo precedente, le metriche seguono la struttura definita al paragrafo 'metriche'. In questo capitolo, la prima parte della struttura sarà:

`M-PRD-[S.I.]`

7.1) Funzionalità

- **M-PRD-CRV** - Copertura Requisiti Vincolanti: percentuale di requisiti vincolanti coperti.

$$\mathrm{CRV} = \frac{\mathrm{RVC}}{\mathrm{RVT}} * 100$$

  - **RVC**: Requisiti Vincolanti Coperti;
  - **RVT**: Requisiti Vincolanti Totali.

- **M-PRD-CRD** - Copertura Requisiti Desiderabili: percentuale di requisiti desiderabili coperti.

$$\mathrm{CRD} = \frac{\mathrm{RDC}}{\mathrm{RDT}} * 100$$

  - **RDC**: Requisiti Desiderabili Coperti;
  - **RDT**: Requisiti Desiderabili Totali.

- **M-PRD-CRO** - Copertura Requisiti Opzionali: percentuale di requisiti opzionali coperti.

$$\mathrm{CRO} = \frac{\mathrm{ROC}}{\mathrm{ROT}} * 100$$

  - **ROC**: Requisiti Opzionali Coperti;
  - **ROT**: Requisiti Opzionali Totali.

7.2) Affidabilità

- **M-PRD-CC** - Code Coverage: percentuale di eseguito durante i test rispetto a quello scritto.

$$\mathrm{CC} = \frac{\mathrm{LCE}}{\mathrm{LCT}} * 100$$

  - **LCE**: Linee di Codice Eseguito;
  - **LCT**: Linee di Codice Totali.

- **M-PRD-BC** - Branch Coverage: percentuale di branch decisionali testati rispetto a quelli creati.

$$\mathrm{BC} = \frac{\mathrm{BT}}{\mathrm{BC}} * 100$$

  - **BT**: Branch Testati;
  - **BC**: Branch creati.

- **M-PRD-SC** - Statement Coverage: percentuale di istruzioni testate rispetto a quelle scritte.

$$\mathrm{SC} = \frac{\mathrm{IE}}{\mathrm{IT}} * 100$$

  - **IE**: Istruzioni Eseguite;
  - **IT**: Istruzioni Totali.

- **M-PRD-FD** - Failure Density: numero di failure riscontrati durante i test.

$$\mathrm{FD} = \frac{F}{T}$$

  - **F**: Numero di Failure;

- **T**: Numero di Test.

7.3) Usabilità

- **M-PRD-FU** - Facilità Di Utilizzo: valuta la complessità di utilizzo del sistema.
- **M-PRD-TA** - Tempo di Apprendimento: valuta il tempo necessario per apprendere il sistema.
- **M-PRD-AU** - Aderenza all'usabilità: misura la capacità del software di aderire alle convenzioni di usabilità nel suo ciclo di vita.
- **M-PRD-AT** - Attrattività: valuta l'aspetto estetico del software.
- **M-PRD-CO** - Comprensibilità: valuta la facilità di comprensione del software.

7.4) Efficienza

- **M-PRD-UR** - Utilizzo Risorse: valuta l'efficienza nell'utilizzo delle risorse.

7.5) Manutenibilità

- **M-PRD-CP** - Complessità Ciclomatica: valuta la complessità del sistema attraverso la misurazione del numero di cammini indipendenti attraverso il grafo di controllo di flusso.
- **M-PRD-CS** - Code Smell: valuta la qualità del codice attraverso la rilevazione di ***Bad Practices***$_G$.
- **M-PRD-MD** - Module Dependency: valuta la dipendenza tra i moduli del sistema

$$\text{MD} = \frac{\text{NDM}}{\text{NTM}} * 100$$

  - **NDM**: Numero di Dipendenze tra Moduli;
  - **NTM**: Numero Totale Moduli.