S.W.A.T.
ENGINEERING

Norme di Progetto

Contatti: swateng.team@gmail.com

Versione: 0.15

Registro delle Modifiche

Versione	Data	Descrizione	Autore	Ruolo
0.15	20-12-2023	Modifiche sezione verifica e creazione sezione controllo di versione e repository	Giacomo D'Ovidio	Amministratore
0.14	12-12-2023	Aggiunta sezione "Diagrammi dei casi d'uso"	Riccardo Toniolo	Amministratore
0.13	12-12-2023	Aggiunta sezione "Metriche per la qualità"	Nancy Kalaj	Amministratore
0.12	06-12-2023	Aggiunte sezioni "Rapporti con la Proponente" "Analisi statica" "Repository"	Nancy Kalaj	Amministratore
0.11	04-12-2023	Aggiunte ulteriore dettaglio a sezioni "Introduzione" "Piano di Progetto" "Convenzioni stilistiche"	Nancy Kalaj	Amministratore
0.10	29-11-2023	Aggiunta sezione "Progettazione" "Codifica" "Gestione Qualità" "Procedure"	Riccardo Costantin	Amministratore
0.9	27-11-2023	Ampliata la sezione riguardante la documentazione	Riccardo Costantin	Amministratore
0.8	26-11-2023	Aggiunta sezione "Fornitura"	Riccardo Costantin	Amministratore
0.7	24-11-2023	Aggiunta sezione "Ruoli progetto"	Riccardo Costantin	Amministratore
0.6	24-11-2023	Aggiunta sezione "Analisi requisiti"	Riccardo Costantin	Amministratore
0.5	17-11-2023	Aggiunta specifica inserimento termini nel Glossario	Matteo Rango	Amministratore
0.4	14-11-2023	Aggiunta sezione "Processi Primari"; Chiarito workflow documentazione e codice	Matteo Rango	Amministratore
0.3	13-11-2023	Aggiunte sezioni "Oggetti Esterni al Repository" e "Tracciamento del Tempo Speso"; modificata organizzazione logica	Matteo Rango	Amministratore
0.2	08-11-2023	Aggiunta sezione "Processi di Supporto" e "Processi Organizzativi"	Matteo Rango	Amministratore
0.1	07-11-2023	Aggiunta sezione "Introduzione"	Matteo Rango	Amministratore

Indice

1 Introduzione

1.1 Scopo del Documento

Il presente documento ha come scopo la definizione delle *best practices*$_G$ e del *way of working*$_G$ che ogni componente del team *SWAT Engineering* ha l'obbligo di rispettare durante l'intero svolgimento del progetto. In questo modo si prova a garantire un metodo di lavoro omogeneo, verificabile e migliorabile nel tempo. La formulazione delle norme avviene in modo progressivo, consentendo al team di apportare aggiornamenti continui in base alle attività e agli strumenti di utilizzo concordati nel corso del lavoro.

1.2 Scopo del progetto

Il progetto "InnovaCity" si concentra sulla creazione di una dashboard intuitiva che permetta a personale amministrativo di monitorare e analizzare il continuo sviluppo di una *smart city*$_G$. L'applicazione comprende una *data pipeline*$_G$ progettata per elaborare dati provenienti da una varietà di simulatori di sensori. Questa pipeline consente la gestione e la visualizzazione ottimale di tali dati, permettendo agli utenti di ottenere rapidamente informazioni rilevanti. L'obiettivo finale è fornire uno strumento per prendere decisioni informate riguardo alla gestione delle risorse della città.

1.3 Glossario

Al fine di evitare possibili ambiguità relative al linguaggio utilizzato nei documenti, viene fornito il *Glossario*, nel quale sono presenti tutte le definizioni di termini aventi un significato specifico che vuole essere disambiguato. Tali termini, sono scritti in *corsivo* e marcati con una $_G$ a pedice. Un'attività che comprende l'inserimento di un termine di glossario può considerarsi conclusa solo nel momento in cui il termine viene scritto e spiegato nel *Glossario*.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato d'appalto C6 - InnovaCity: https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C6.pdf

1.4.2 Riferimenti informativi

- Documentazione git: https://git-scm.com/docs
- Documentazione GitHub: https://docs.github.com/en
- Materiale didattico del corso Metodi e Tecnologie per lo Sviluppo Software 2022/2023: https://stem.elearning.unipd.it/course/view.php?id=5359

  - Lezione 4: GIT;
  - Laboratorio 2: GitHub Version Control System.
- Documentazione Typst: https://typst.app/docs

2 Processi Primari

2.1 Fornitura

2.1.1 Descrizione e scopo

Il processo di fornitura si propone di dettagliare le attività del fornitore per comprendere e soddisfare le richieste della Proponente. Dopo la completa comprensione delle esigenze, il fornitore, in collaborazione con la Proponente, stabilisce tramite contratto la data di consegna del prodotto.
Successivamente, viene redatto il Piano di Progetto per pianificare dettagliatamente le varie attività da svolgere, garantendo un chiaro processo di sviluppo del prodotto finale. L'obiettivo principale è soddisfare in modo chiaro le richieste della Proponente, evitando possibili incomprensioni attraverso una collaborazione continua.

Il processo si articola in diverse fasi:

- Definizione chiara dei requisiti soddisfatti dal prodotto finale;
- Contrattazione, con l'obiettivo di:
  1. Richiesta di incontri di formazione sulle tecnologie consigliate per ottimizzare lo sviluppo;
  2. Ricezione di un feedback approfondito sull'utilizzo delle tecnologie proposte.
- Pianificazione: consiste nell'individuare preventivamente una suddivisione precisa di tutte le ore produttive disponibili, seguita da una stima dei costi per ciascun incremento di lavoro (sprint);
- Esecuzione: la progettazione e il conseguente sviluppo del prodotto procedono di pari passo con la stesura della documentazione relativa al progetto;
- Continuo controllo e verifica;
- Completamento e consegna.

2.1.2 Rapporti con la Proponente

La Proponente si è resa disponibile attraverso vari canali, come e-mail, *Google Meet* e *Element*, per stabilire una comunicazione frequente e risolvere prontamente eventuali dubbi o domande che possono emergere durante lo svolgimento del progetto. Sin dall'inizio è stato concordato di organizzare incontri regolari, in particolare al termine di ciascuno sprint, fissati per il venerdì alle 10:30. Questi incontri assumono la forma di una sessione di *Sprint Review*, in cui il team esibisce quanto prodotto e ottiene feedback sull'andamento del lavoro.

Gli incontri con la Proponente si suddividono principalmente in tre categorie:

- Incontri di formazione: utilizzati per acquisire familiarità con nuove tecnologie, approfondire determinati concetti o migliorare competenze specifiche richieste dal progetto;
- Incontri di analisi dei requisiti: mirati a chiarire, discutere e validare i requisiti del progetto, garantendo una comprensione chiara e condivisa tra il team e la Proponente;
- Sprint Review: fase conclusiva di ogni sprint durante la quale vengono presentati i risultati ottenuti, con l'intento di ricevere feedback dalla Proponente.

Ogni incontro con la Proponente viene sintetizzato e documentato nel Verbale Esterno di riferimento. Il verbale viene successivamente presentato alla Proponente per essere validato tramite firma, ottenendo così un'approvazione formale del resoconto delle discussioni avvenute all'incontro.

2.1.3 Documentazione fornita

Di seguito si elencano i documenti che il team *SWAT Engineering* si impegna a consegnare ai Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin e alla Proponente Sync Lab.

2.1.3.1 Analisi dei Requisiti

L'analisi dei requisiti, svolta dagli Analisti, costituisce una fase cruciale nello sviluppo del sistema software. Il suo obiettivo principale è definire in dettaglio le funzionalità necessarie affinché il prodotto soddisfi pienamente le richieste della Proponente. Il documento omonimo comprende una serie di elementi essenziali:

- Descrizione e scopo del prodotto;
- **Definizione degli attori**: entità o persone che interagiscono con il *sistema*$_G$
- **Definizione dei casi d'uso**: rappresentazione narrativa di scenari specifici che descrivono come gli attori interagiscono con il sistema. I casi d'uso offrono una visione chiara delle azioni che possono essere eseguite all'interno del sistema e come gli utenti interagiscono con esso; un elenco preciso di azioni intraprese dall'attore per provocare un determinato caso d'uso viene fornito all'interno dello stesso, al fine di facilitare l'estrazione dei requisiti corrispondenti;
- **Definizione di requisiti**: individuazione dei requisiti obbligatori e desiderabili e loro categorizzazione in:
  - Requisiti funzionali: specificano le operazioni che il sistema deve essere in grado di eseguire;

- Requisiti di qualità: riguardano gli aspetti di qualità del software, inclusi gli attributi prestazionali, affidabilità, usabilità e sicurezza;
- Requisiti di vincolo: delineano vincoli e limitazioni che il sistema deve rispettare. Possono includere restrizioni tecnologiche, normative o di risorse.

2.1.3.2 Piano di Progetto

Documento redatto dal Responsabile che offre una visione dettagliata dell'intero processo di sviluppo di un progetto. Esso fornisce al team una guida approfondita per mantenere l'allineamento con gli obiettivi, gestire le risorse in modo efficace e affrontare le sfide che potrebbero emergere durante lo sviluppo del progetto.

È composto da:

- **Analisi dei rischi**: identifica, valuta e gestisce i potenziali rischi che potrebbero influenzare il successo del progetto. Questo processo è essenziale per consentire al team di adottare misure preventive. I rischi vengono categorizzati in rischi tecnologici, di comunicazione e di pianificazione. Per ogni rischio, si compie un analisi mirata per individuarne i segnali di manifestazione, valutarne l'occorrenza e la pericolosità, pianificare strategie di mitigazione e valutare l'efficacia di tali strategie nel contenere gli effetti del rischio preso in esame;
- **Modello di sviluppo**: approccio metodologico che viene scelto per guidare il processo di sviluppo del prodotto. Esso definisce la struttura di lavoro che viene seguita per l'intero ciclo di vita del progetto. Con l'adozione del modello di sviluppo agile Scrum, questa sezione descrive le pratiche e gli eventi principali del *framework*, illustrando il modo in cui vengono utilizzati ed implementati dal team;
- **Pianificazione**: fornisce una roadmap dettagliata delle attività, delle risorse, e delle scadenze associate al progetto. In particolare, vengono pianificate le attività necessarie per raggiungere gli obiettivi previsti per ogni sprint, dall'inizio sino al termine del progetto, e la loro distribuzione temporale;
- **Preventivo**: basandosi sulla pianificazione eseguita a priori, determina la ripartizione delle ore produttive a disposizione di ogni componente del team nei vari ruoli per ogni sprint. Questo assicura il conseguimento degli obiettivi prefissati e un utilizzo oculato delle risorse. La suddivisione delle ore determina altresì il costo preventivato per ogni sprint;
- **Consuntivo**: partendo dalla rendicontazione delle ore produttive impiegate da ciascun membro del team eseguita a posteriori, determina la ripartizione

effettiva delle ore osservata durante lo sprint e, di conseguenza, anche il costo effettivo. Inoltre, si conduce una breve analisi retrospettiva per giustificare le scelte effettuate nel preventivo, evidenziare eventuali discostamenti e delineare i cambiamenti nella strategia utilizzata per pianificare il resto del lavoro, qualora dovessero rendersi necessari.

2.1.3.3 Piano di Qualifica

Documento che descrive le strategie e gli approcci adottati per garantire la qualità del prodotto o del servizio che si sta sviluppando. Il suo scopo principale è quello di definire le modalità di verifica e validazione, nonché gli standard e le procedure di qualità che verranno seguite durante l'intero ciclo di vita del progetto.

È caratterizzato da:

- **Qualità di processo**: standard e procedure adottate per garantire la qualità durante lo sviluppo del progetto. Informazioni sulle attività di gestione della qualità, le metodologie utilizzate, e come vengono misurati e migliorati i processi stessi;
- **Qualità di prodotto**: standard, specifiche e caratteristiche che il prodotto deve soddisfare per essere considerato di qualità. Include anche le metriche e i criteri di valutazione utilizzati per misurare la qualità del prodotto;
- **Specifiche dei test**: specifiche dettagliate dei test che verranno condotti durante lo sviluppo del progetto;
- **Resoconto e valutazioni**: resoconto delle attività svolte e delle valutazioni effettuate durante il progetto. Utili per capire come il progetto si sta sviluppando rispetto agli obiettivi e alle aspettative, e per identificare eventuali azioni correttive necessarie.

2.1.3.4 Glossario

Il Glossario funge da catalogo esaustivo che raccoglie i termini tecnici impiegati all'interno del progetto, offrendo definizioni chiare e precise. Questo documento non solo previene fraintendimenti e ambiguità ma favorisce anche una comprensione condivisa della terminologia propria del settore, migliorando la coerenza e la qualità della documentazione prodotta. Inoltre, contribuisce a garantire un allineamento efficace tra tutti i partecipanti al progetto, consentendo un flusso di lavoro più efficiente e risultati più accurati.

2.1.3.5 Lettera di Presentazione

La Lettera di Presentazione accompagna la consegna del prodotto software e della documentazione pertinente ad ogni revisione e sottolinea l'impegno che

il team *SWAT Engineering* si assume nel completare e consegnare il prodotto entro le scadenze concordate. Inoltre, espone un preventivo aggiornato rispetto a quello presentato in occasione dell'ultima revisione, dove costo e data di consegna del progetto vengono analizzati rispetto all'andamento corrente e confermati, o eventualmente modificati, in seguito a scelte ragionate.

2.1.4 Strumenti

Gli strumenti adottati per agevolare il processo di fornitura sono i seguenti:

2.1.4.1 Google Calendar

Strumento di calendario, utilizzato dal team per gestire impegni e attività in modo organizzato, oltre che per condividere eventi con la Proponente.

2.1.4.2 Google Slides

Servizio di creazione di presentazioni multimediali, utilizzato dal team per assemblare i diari di bordo.

2.1.4.3 Google Sheets

Servizio di creazione di *spreadsheet*$_G$, utilizzato dal team per la rendicontazione delle ore produttive impiegate da ciascun componente nel corso di uno sprint.

2.1.4.4 Google Meet

Servizio per creare e partecipare a videochiamate, utilizzato dal team per gli incontri telematici con la Proponente.

2.1.4.5 Online Gantt

Online software per creare diagrammi di Gantt, utilizzato dal Responsabile per delineare la distribuzione temporale delle attività pianificate per ogni sprint nella sezione di **Pianificazione** del Piano di Progetto.

2.1.4.6 Draw.io

Software per creare diagrammi e grafici di varia natura, utilizzato dagli Analisti per creare i diagrammi UML dei casi d'uso nella sezione **Casi d'uso** (Sezione 2.2.2.2) dell'Analisi dei Requisiti.

2.2 Sviluppo

2.2.1 Descrizione e scopo

Il processo di sviluppo rappresenta la serie di attività che il team *SWAT Engineering* deve svolgere affinché, successivamente, riesca

nell'implementazione del prodotto software, rispettando i requisiti e le date di scadenza concordate con la Proponente. In particolare, si ha:
- **Analisi dei Requisiti**;
- **Progettazione**;
- **Codifica**.

2.2.2 Analisi dei Requisiti

2.2.2.1 Descrizione e scopo
L'Analisi dei Requisiti viene redatta dagli Analisti e contiene:
- **Introduzione**: esplicita lo scopo del documento, lo scopo del prodotto e i riferimenti utilizzati;
- **Descrizione**: esplicita le funzionalità attese del prodotto e le caratteristiche degli utenti;
- **Casi d'uso**: individua gli attori e tutte le interazioni che possono avere con il sistema;
- **Requisiti**: le caratteristiche da soddisfare e le fonti da cui sono state estratte.

2.2.2.2 Identificazione dei casi d'uso
I casi d'uso sono identificati nel seguente modo:

UC[Numero].[Numero sottocaso] [Titolo]

legenda:
- **Numero**: numero del caso d'uso;
- **Numero sottocaso**: numero del sottocaso del caso d'uso generale se presente;
- **Titolo**: breve titolo che descrive il contesto del caso d'uso in questione.

2.2.2.3 Struttura dei casi d'uso
I casi d'uso sono strutturati nel seguente modo:

- Attore;
- Precondizioni;
- Postcondizioni;
- Scenario Principale;
- Scenari Secondari (ove necessario);
- Estensioni, se presenti;
- Specializzazioni, se presenti.

2.2.2.4 Requisiti
I requisiti trovati vengono classificati nei seguenti modi:

- **Funzionali**:

  un requisito funzionale specifica una funzionalità che il sistema deve essere in grado di svolgere;

- **Qualità**:

  un requisito di qualità stabilisce gli standard e i criteri che il sistema deve soddisfare per garantire prestazioni, affidabilità, sicurezza e altri aspetti correlati alla qualità;

- **Vincolo**:

  un requisito di vincolo è una restrizione o una condizione imposta al progetto.

2.2.2.5 Identificazione dei requisiti

I requisiti trovati hanno un codice univoco con la seguente sintassi:

R[Importanza][Tipo][Numero]

legenda:

- **Importanza**:
  - O -> se requisito obbligatorio;
  - D -> se requisito desiderabile;
  - P -> se requisito opzionale.
- **Tipo**:
  - F -> se funzionale;
  - Q -> se di qualità;
  - V -> se di vincolo.
- **Numero**: per ogni requisito aggiunto il numero viene incrementato.

2.2.2.6 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.1.2.

Metrica	Descrizione
ROS	Requisiti Obbligatori Soddisfatti
RDS	Requisiti Desiderabili Soddisfatti
ROPS	Requisiti Opzionali Soddisfatti

2.2.3 Progettazione

2.2.3.1 Descrizione e scopo

L'attività di progettazione è affidata ai Progettisti, i quali devono definire le caratteristiche del prodotto finale basandosi sui requisiti specificati nel

documento *Analisi dei Requisiti*. La fase di progettazione segue l'analisi dei requisiti, dove sono definite le necessità e le aspettative per il prodotto. I Progettisti traducono queste informazioni in una struttura architetturale definita, organizzando il sistema in componenti specifici e definendo le interazioni tra di essi. In questo modo, la progettazione costituisce un passo essenziale nel percorso di sviluppo, contribuendo a trasformare i requisiti in un piano tangibile per la creazione del prodotto finale.

Si definiscono tre sottoattività:

1. ***Technology Baseline***: implica la selezione e la definizione delle tecnologie di base utilizzate per la realizzazione del sistema. Questo comprende decisioni relative a linguaggi di programmazione, librerie e *framework*. Tale processo porta alla creazione di un Proof of Concept (PoC_G);

   Include:
   - **Proof of Concept**: consiste nella creazione di una versione parziale del prodotto, includendo alcune delle funzionalità stabilite durante l'analisi dei requisiti. L'obiettivo è valutare la fattibilità del prodotto completo;
   - **scelte tecnologiche**: consiste nello stabilire quali tecnologie adottare per lo sviluppo del Poc, anche su consiglio della Proponente.
2. ***Progettazione Architetturale***: definizione ad alto livello dell'architettura del sistema; si concentra sulla suddivisione del sistema in componenti e moduli, definendo le relazioni tra di essi e specificando le linee guida per l'organizzazione e l'interazione delle varie parti;
3. ***Product Baseline***: segna un punto stabile nel processo di progettazione, in cui le specifiche tecniche, le funzionalità principali e l'architettura del prodotto sono definite in modo dettagliato e accettate dalle parti coinvolte. Include tutti gli elementi essenziali e i requisiti chiave del prodotto che devono essere soddisfatti, fornendo una base solida per lo sviluppo continuo del prodotto. Questo processo porta infine alla realizzazione di un *Minimum Viable Product* (MVP_G); Include:
   - *Design Patterns*$_G$
   - Definizione delle classi;
   - Diagrammi UML che includono:
     - classi;
     - *package*;
     - sequenze: utilizzato per descrivere uno scenario che costituisce una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate;
     - attività: diagramma comportamentale che illustra il flusso delle attività attraverso un sistema.

- Test di Unità su ogni componente.

2.2.3.2 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.1.2.

Metrica	Descrizione
SFIN	Structure Fan In
SFOUT	Structure Fan Out
ATC	Attributi per Classe
PM	Parametri per Metodo
LCM	Linee di Codice per Metodo

2.2.3.3 Diagrammi UML dei casi d'uso

Un diagramma dei casi d'uso rappresenta uno strumento di modellazione ampiamente impiegato per documentare e delineare le funzionalità di un sistema. La sua utilità risiede nel tracciare i flussi operativi attraverso una rappresentazione visiva, descrivendo il modo in cui un utente interagisce con il sistema.

Gli scenari d'uso sono organizzati in sequenze di azioni, illustrando le operazioni necessarie per consentire a un utente di portare a termine una specifica attività (realizzare uno scopo), e sono interconnessi mediante linee. Questo tipo di diagramma risulta particolarmente prezioso nella progettazione di sistemi, in quanto offre un'illustrazione rapida e intuitiva delle dinamiche di lavoro e delle interazioni tra l'utente e il sistema.

È fondamentale notare che la rappresentazione fornita dai diagrammi dei casi d'uso non si addentra nei dettagli implementativi, poiché il loro scopo principale è descrivere la funzionalità, considerandola come un elemento esterno al sistema.

I diagrammi dei casi d'uso sono composti da:

- **Attore**: rappresenta un agente esterno coinvolto nelle interazioni con il sistema. Si tratta di una qualsiasi entità in grado di interagire con il sistema, infatti ogni caso d'uso determina una funzionalità che viene messa a disposizione di tale attore, tuttavia, senza entrare nei dettagli implementativi.

  A livello di diagramma, l'attore è simboleggiato da un' icona umana stilizzata, identificabile mediante un'etichetta univoca e rappresentativa, posizionata al disotto di tale figura.

Figure 1: Figura rappresentante un attore.

- **Caso d'uso**: delinea le operazioni eseguibili dall'utente sul sistema. Un singolo caso d'uso si compone di una breve esposizione delle funzioni messe a disposizione del sistema per uno o più utenti nell'ambito di un software. In modo specifico, offre una descrizione dettagliata del comportamento dell'utente mentre interagisce con il software.

  Generalmente, un caso d'uso è costituito da una sequenza di situazioni che esplicitano le diverse eventualità che possono manifestarsi durante l'interazione tra l'utente e il software. La sua rappresentazione comprende un'identificazione univoca, espressa come UCx.y (dove x indica il numero del caso d'uso, e y indica il fatto che stiamo trattando un eventuale sotto-caso d'uso del caso d'uso UCx), seguita da una concisa ma completa descrizione della funzione stessa.

  Ogni caso d'uso discute i seguenti punti:
  - **Attore principale**: l'attore che intende compiere lo scopo rappresentato dal caso d'uso;
  - **Precondizioni**: stato in cui il sistema si deve trovare prima dell'avvio della funzionalità rappresentata dal caso d'uso;
  - **Postcondizioni**: stato in cui il sistema si troverà dopo che l'utente avrà portato a termine lo scopo rappresentato dal caso d'uso;
  - **Scenario principale**: descizione accurata della funzionalità rappresentata dal caso d'uso;
  - **Specializzazioni**: nel caso di uno use case generale, vengono indicati i codici dei casi d'uso che lo specializzano;
  - **Inclusioni**: vengono specificati i codici dei casi d'uso che vengono inclusi nel caso d'uso trattato;
  - **Estensioni**: vengono specificati i codici dei casi d'uso che rappresentano scenari secondari.

  Ciascun caso d'uso è connesso, attraverso una linea continua, agli attori che hanno autorizzazioni per accedere a quella particolare funzione.

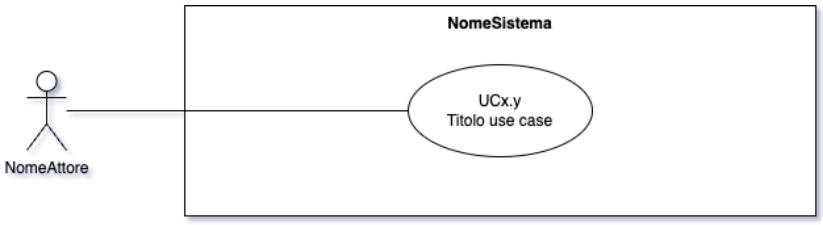

Figure 2: Figura rappresentante un caso d'uso.

In ogni diagramma dei casi d'uso possono essere definite:

- **Generalizzazioni**: il concetto di generalizzazione può essere esteso sia agli attori che ai casi d'uso. La generalizzazione di un attore si verifica quando un attore di livello superiore, dotato di abilità più generiche, viene specializzato in comportamenti più specifici nei sottostanti attori. Ogni attore sottostante eredita le funzionalità dal suo attore padre, integrandole con ulteriori aspetti rilevanti al proprio contesto.

  Per quanto riguarda i casi d'uso, i casi figli hanno la possibilità di aggiungere o modificare il comportamento dei casi d'uso ereditati dal caso padre. Tutte le funzionalità non ridefinite nei casi figlio mantengono la definizione ereditata. La generalizzazione degli attori e dei casi d'uso è simboleggiata da una freccia continua con triangolo vuoto bianco, che si estende da un elemento figlio a un elemento padre.

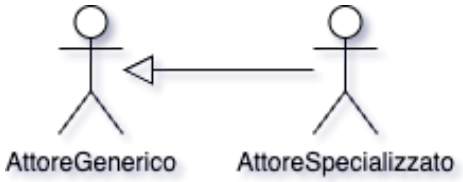

Figure 3: Figura rappresentante una generalizzazione tra attori.

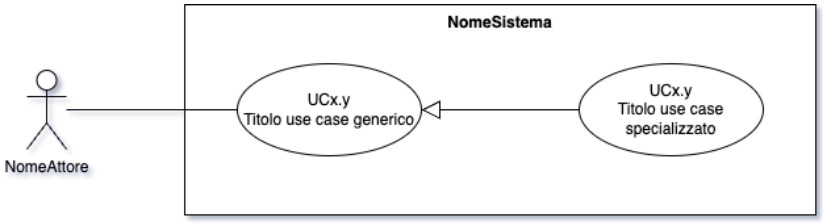

Figure 4: Figura rappresentante una generalizzazione tra casi d'uso.

- **Inclusioni**: supponiamo che vi sia una relazione di inclusione tra un caso d'uso A e un caso d'uso B se ogni istanza del caso d'uso A deve necessariamente eseguire le istanze del caso d'uso B. Questo assegna al caso d'uso A la responsabilità di eseguire il caso d'uso B, eliminando la duplicazione e favorendo il riutilizzo di una struttura comune. La connessione di inclusione viene simboleggiata da una freccia tratteggiata

che collega il caso d'uso A a tutti i casi d'uso inclusi, come nel caso del caso d'uso B nell'esempio. Sopra la freccia verrà annotata la direttiva "include".

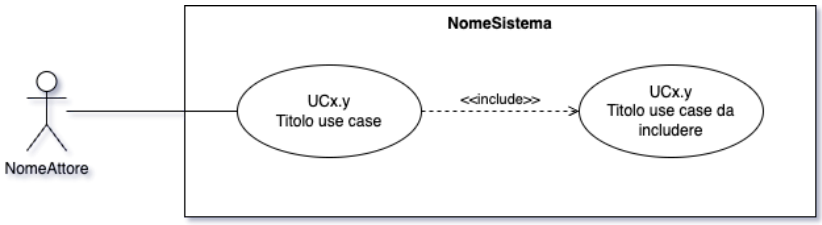

Figure 5: Figura rappresentante un' inclusione tra casi d'uso.

- **Estensioni**: nel contesto dei diagrammi dei casi d'uso UML, la relazione di estensione indica una connessione tra due casi d'uso, A e B, segnalando che ogni istanza del caso d'uso A può condizionalmente eseguire anche il caso d'uso B. L'esecuzione del caso d'uso B avviene soltanto in specifiche circostanze o sotto condizioni particolari durante l'esecuzione del caso d'uso A, interrompendo temporaneamente il flusso del caso d'uso A. La responsabilità dell'esecuzione del caso d'uso esteso (B) ricade su chi estende (nel caso, il caso d'uso B). Questa relazione viene visualizzata graficamente con una freccia tratteggiata dal caso d'uso esteso (B) al caso d'uso base (A), con l'etichetta "extend".

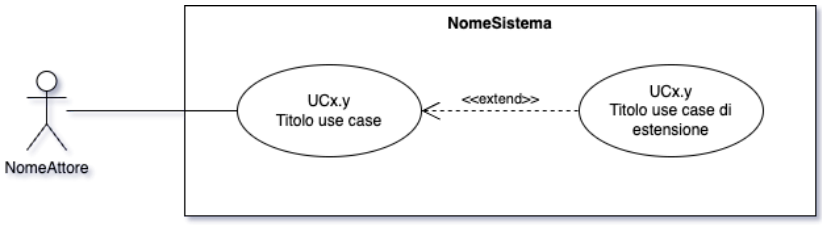

Figure 6: Figura rappresentante un' estensione tra casi d'uso.

2.2.4 Codifica

2.2.4.1 Descrizione e scopo

L'attività di codifica viene svolta dai Programmatori, i quali sono responsabili della traduzione delle decisioni progettuali nel codice sorgente. I Programmatori seguono le linea guida e le *best practices* stabilite durante la fase di progettazione architetturale.

2.2.4.2 Aspettative

Ci si aspetta che il codice sviluppato rispetti determinate caratteristiche:

- Conformità alle specifiche;
- Chiarezza e comprensibilità;
- Ottimizzazione delle prestazioni;
- Supplemento di test per verificare la correttezza e il funzionamento.

2.2.4.3 Stile di codifica

[...]

2.2.4.4 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.1.2.

Metrica	Descrizione
CCM	Complessità Ciclomatica per Metodo
CC	Code Coverage

3 Processi di Supporto

3.1 Documentazione

3.1.1 Descrizione e scopo

La documentazione è l'insieme di informazioni rappresentate sotto forma di testo scritto che accompagna un prodotto software, svolgendo un ruolo essenziale nella descrizione del prodotto per coloro che lo sviluppano, lo distribuiscono e lo utilizzano. Il suo obiettivo primario è facilitare l'attività di sviluppo ai membri del team durante l'intero ciclo di vita del progetto e garantirne la coerenza,tracciando tutti i processi e le attività coinvolte per migliorare la qualità del risultato finale e semplificare la manutenzione. L'implementazione di regole chiare e di una struttura uniforme non solo migliora la fruibilità e la comprensione, ma favorisce anche la collaborazione all'interno del team, contribuendo in modo significativo al successo complessivo del progetto software.

3.1.2 Lista Documenti

I documenti che verranno prodotti sono:

- *Norme di Progetto*;
- *Piano di Progetto*;
- *Piano di Qualifica*;
- *Analisi dei Requisiti*;
- *Glossario*;
- *Verbali*:
  1. *Interni*;
  2. *Esterni*.

3.1.3 Ciclo di vita dei documenti

Ogni documento segue le fasi del seguente *workflow*$_G$:

1. Si crea un branch per lo sviluppo del documento nell'apposita repository *Docs* e si mette in uso.
2. Si copia dall'apposita repository `Templates` il template relativo al file che si deve redigere, e lo si inserisce nella cartella appropriata.
3. Si redige il documento o una sua sezione. Nel caso di documenti nuovi, in cui è necessario un elevato parallelismo di lavoro, è possibile usare Google Drive per la prima stesura e successivamente caricare il documento all'interno del branch.
4. Nel file `changelog.typ` si aggiunge una riga **in coda**, secondo il seguente formato: `<versione>,<data-modifica>,<descrizione-modifica>,<nome-`

`autore>,<ruolo-autore>`; la versione segue le regole descritte nella Sezione 3.1.6.
5. Si esegue la commit sul branch creato.
6. Si apre una pull request dal branch appena creato verso il branch `main`: se il documento non è pronto per la verifica, ma ha bisogno di ulteriori modifiche, si apre la pull request in modalità `draft`, altrimenti in modalità normale, spostando la issue nell'apposita corsia di stato "Ready to Review".
7. Per ulteriori modifiche richieste dal/dai verificatore/i si ripetono i punti, **in ordine**, dal punto *3* al punto *5*.
8. Si elimina, **solo quando la pull request viene chiusa o risolta**, il branch creato.

La modifica di un documento avviene allo stesso modo, saltando il passo 2. Ogni cambiamento di stato è accompagnato dal conseguente movimento della issue, associata allo sviluppo, attraverso le diverse corsie dell'issue tracking system.

3.1.4 Template Typst

Per la stesura dei documenti viene usato un template in formato *Typst*. Il template fornisce una struttura e un formato predefinito per semplificare la creazione di documenti. Serve a garantire coerenza, risparmiare tempo, standardizzare la presentazione e contribuire a una produzione di documenti più efficiente e professionale. Sono stati sviluppati quattro template distinti per adattarsi alle diverse esigenze di documentazione:
- documentazione ufficiale;
- lettere di presentazione;
- verbali per incontri interni ed esterni.

Ogni template è progettato per garantire coerenza e facilità d'uso, con piccole modifiche per rispecchiare le specificità di ciascun tipo di documento.

3.1.5 Nomenclatura

La consueta nomeclatura per i documenti si ottiene unendo, attraverso un underscore (_), il nome del file in *CamelCase* senza spazi (`NomeDelFile`) e la sua versione (`3.5`). Ad esempio `NormeDiProgetto_2.6.pdf`. Nel caso di documenti il cui nome contiene una data, essa si inserisce dopo il nome, ma prima della versione, sempre separandolo con gli underscores, nella forma `ggmmaa` senza separatori tra i singoli componenti della data: `gg` rappresenta il giorno, sempre scritto in due cifre, allo stesso modo `mm` rappresenta il mese, mentre l'anno è rappresentato da `aa`, corrispondente alle ultime due cifre dell'anno corrente.

3.1.6 Versionamento

Il versionamento avviene secondo il seguente formato **x.y**:

- **y** si incrementa una volta effettuata una modifica e la sua conseguente verifica;
- **x** si incrementa quando si effettua la modifica definitiva in vista di una verifica di avanzamento, questo comporta l'azzeramento di **y**.

Due modifiche, fatte in momenti diversi, differiscono l'una dall'altra solo se hanno scopi diversi. Ad esempio non è necessario incrementare la versione se viene fatta una modifica alla stessa sezione in due giorni differenti; anche se faccio una modifica, ed essa non viene approvata, non è necessario incrementare la versione con le nuove modifiche proposte dal/dai verificatore/i, dal momento che modifica e verifica "viaggiano" parallelamente.

3.1.7 Struttura

3.1.7.1 Prima Pagina

- **logo team**: situato in alto a destra;
- **Titolo**:
  - Nome del documento, se diverso dai verbali;
  - per i verbali interni: Verbale Interno;
  - per i verbali esterni: Verbale Esterno;
- **Data**: solo se si tratta di verbali, vedere Sezione 3.1.8.2;
- **Contatti**: l'email del team;
- **Versione**: l'ultima versione del documento;
- **logo università**: in basso a destra.

3.1.7.2 Intestazione

Su ogni pagina del documento, eccetto la prima, si trova il titolo del documento seguito dalla sua versione e il logo del team.

3.1.7.3 Registro delle modifiche

Tabella con l'intestazione:

- **Versione**: versione del documento;
- **Data**: data della modifica apportata;
- **Descrizione**: cosa è stato modificato o aggiunto al file;
- **Autore**: l'autore della modifica;
- **Ruolo**: ruolo dell'autore al momento della modifica.

3.1.7.4 Indice
In una nuova pagina deve essere presente l'indice, utile per facilitare la ricerca e la navigazione all'interno del documento.

3.1.7.5 Verbali
I verbali differiscono leggermente da un documento ufficiale, in quanto non evolvibili nel tempo.

Si compongono principalmente di 2 sezioni:
- **Partecipanti**: si indica data di inizio e fine incontro e il luogo in cui si è svolto. A seguire, i nomi dei partecipanti del team e la durata della presenza di ciascuno vengono rappresentati in forma tabellare. Se il verbale è esterno si indicano anche i partecipanti della Proponente;
- **Sintesi dell'incontro**: Riassunto degli argomenti trattati durante la riunione.

Il verbale esterno oltre alle sezioni sopra elencate ha una pagina per la convalida, attraverso firma, del documento.

3.1.8 Convenzioni stilistiche
- **Grassetto**:
  - Titoli di sezioni/sottosezioni/paragrafi di un documento;
  - Parole a cui si vuole dare enfasi;
  - Definizioni di termini negli elenchi puntati.
- **Corsivo**:
  - I nomi dei documenti;
  - I termini di glossario (seguiti da $_{G}$).
- **Caratteri maiuscoli**:
  - Le iniziali dei nomi;
  - Le lettere che compongono un acronimo;
  - Le iniziali dei ruoli svolti dai componenti del gruppo;
  - L'iniziale del termine "Proponente".

3.1.8.1 Elenchi puntati
Le voci di ogni elenco iniziano con lettera maiuscola e terminano con punto e virgola ';', eccetto l'ultima voce che termina con punto normale '.'.

3.1.8.2 Formato delle date
Viene adottato il formato "DD-MM-YYYY":
- DD: giorno con 2 cifre;
- MM: mese con 2 cifre;
- YYYY: anno con 4 cifre.

3.1.9 Strumenti

Il gruppo utilizza:

- **Typst**: linguaggio di markup utilizzato per la redazione di documenti, noto per la sua semplicità e flessibilità nella formattazione di testi strutturati;
- **Visual Studio Code**: un popolare ambiente di sviluppo integrato (IDE), noto per la sua leggerezza, versatilità e la vasta gamma di estensioni che permettono la personalizzazione e offrono supporto per numerosi linguaggi di programmazione;
- **GitHub**: una piattaforma di hosting per progetti di sviluppo software basati su Git. Fornisce un sistema di controllo delle versioni distribuito e strumenti per la gestione del codice sorgente, delle issue e delle pull request, facilitando la collaborazione all'interno di un team di sviluppo.

3.1.10 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.1.3.

Metrica	Descrizione
IG	Indice Gulpease
CO	Correttezza Ortografica

3.2 Gestione della Configurazione

3.2.1 Descrizione e scopo

3.2.2 Issue Tracking System

Come ITS si utilizza *Github* che, attraverso le funzioni di *Project*, *Issues* e *Pull requests*, garantisce una struttura all'organizzazione di progetto.

Le *Corsie di Stato*$_G$ descrivono lo stato attuale delle attività, all'interno del *Project* nell'*ITS* sono presenti:
Backlog attività individuate da svolgere.
Ready attività individuate per il completamento durante il prossimo *sprint*$_G$.
In Progress attività che sono in corso d'opera da parte dei redattori.
Ready to Review attività svolte che sono pronte per essere verificate.
In Review attività in corso di verifica da parte dei verificatori.
Done attività le cui modifiche sono state verificate e accettate.

3.2.3 Strumento di Condivisione

Per la condivisione veloce o la creazione di bozze si utilizza *Google Drive*. Uno dei suoi principali casi d'uso consiste nella collaborazione in tempo reale nella stesura di sezioni testuali ampie, da inserire successivamente nella

documentazione (questo risulta particolarmente utile nel momento in cui il documento è alla sua prima stesura). Viene inoltre utilizzato come sistema per l'immagazzinamento di conoscenze acquisite durante lo svolgimento del progetto.

3.2.4 Tracciamento del Tempo Speso

Al fine di tracciare il tempo speso nel corso del progetto, nei diversi ruoli, si utilizza uno spreadsheet appositamente creato, disponibile all'interno di Google Drive. A fine giornata, ogni membro del team inserisce le proprie ore **produttive** svolte quel giorno, secondo la sua miglior stima del rapporto tra ore di orologio e ore produttive. Si inserisce una sola riga per ogni giornata e nella descrizione si aggiungono brevi titoli rappresentativi delle attività svolte.

3.2.5 Controllo di versione e repository

Come sistema di controllo di versione si utilizza *Git*.

Vengono utilizzate le seguenti *repository*:

- https://github.com/SWATEngineering/Docs: questa repository viene impiegata dal team per condividere e revisionare il codice sorgente legato alla documentazione del progetto. Viene utilizzata per collaborare, revisionare e mantenere aggiornati i documenti di lavoro, consentendo ai membri del team di contribuire e gestire in modo efficiente la documentazione;
- https://github.com/SWATEngineering/SWATEngineering.github.io: questa repository funge da piattaforma per i documenti compilati e approvati. Il sito web https://swatengineering.github.io/ rappresenta la versione web della documentazione finale e approvata dal team. È utilizzata per presentare al pubblico una visione consolidata dei documenti di progetto;
- https://github.com/SWATEngineering/InnovaCity: questa repository è dedicata alla condivisione e revisione del codice sorgente relativo al prodotto software "InnovaCity". È qui che il team lavora e collabora sul codice del prodotto stesso, consentendo una gestione centralizzata del lavoro di sviluppo e delle modifiche apportate al software.

All'interno dei repository *Docs* e *Innovacity* si utilizza il *Rebase workflow*$_G$ come metodo di gestione, con l'utilizzo dei *Feature branch* per separare logicamente il lavoro da svolgere. Nel primo repository però, i *Feature branch* si derivano direttamente dal *main*, mentre nel secondo si derivano dal branch *dev*. Questo impone quindi che, prima di andare ad effettuare la chiusura di una *Pull request*, si vada ad effettuare un *rebase* del branch di derivazione, per rendere il nostro branch di sviluppo aggiornato rispetto alla base.

I *Feature branch* vengono aperti a partire dalle issue create nell' *Issue Tracking System* (vedi Sezione 3.2.2). Si procede poi ad associare una *Pull request*, a una o più issue collegate tra loro, per effettuare la verifica.

Nel caso del repository *InnovaCity*, il branch *main* viene utilizzato per la pubblicazione di cambiamenti *major*, ovvero quando sono state implementate diverse funzionalità significative che contribuiscono all'avanzamento del progetto. In questa circostanza, è compito del Responsabile eseguire l'approvazione finale.

3.3 Verifica

3.3.1 Descrizione e Scopo

Qualsiasi processo istanziato durante lo svolgimento del progetto, prima di potersi considerare completato, dev'essere sottoposto a verifica. Lo scopo primario di questo processo è garantire la correttezza dei prodotti e la loro adesione ai vincoli di qualità individuati ed elencati all'interno del documento *Piano di Qualifica*. Il *Piano di Qualifica* funge da punto di riferimento per il Verificatore: in esso vengono fornite tutte le linee guida a cui il Verificatore deve aderire garantendo uniformità e coerenza al processo di verifica e garantendone la ripetibilità.

3.3.2 Strumenti

Gli strumenti adottati per agevolare il processo di verifica sono i seguenti:

3.3.2.1 GitHub

GitHub offre una funzionalità di *review* all'interno del meccanismo di *pull request*, permettendo al Verificatore di visualizzare facilmente le ultime modifiche apportate al prodotto. Il Verificatore inserisce commenti specifici che indicano le correzioni o le migliorie necessarie e, al termine della review, la invia richiedendo le modifiche indicate; in seguito all'intervento correttivo dell'autore, il Verificatore esegue ulteriori revisioni fino a quando il documento non presenta più errori e rispetta i criteri di conformità richiesti. A seguito del processo di verifica il Verificatore si occupa di spostare la issue nella corsia di stato adeguata all'interno della *Kanban Board*:

- *done* qualora la revisione abbia avuto esito positivo;
- *in progress* in caso siano richieste modifiche.

L'utilizzo delle review in GitHub non solo facilita il tracciamento del processo di verifica, ma consente anche al team di accedere e consultare facilmente l'intera cronologia del codice o del documento di interesse all'interno della *repository*$_G$ del progetto. Questo approccio garantisce un processo di verifica

trasparente, tracciabile e conforme alle linee guida stabilite. Inoltre GitHub impedisce l'unione dei rami oggetto di pull request fino a quando l'ultimo commit non viene verificato e approvato. Ciò garantisce che ogni prodotto, che viene integrato al ramo principale, sia effettivamente revisionato da almeno un membro del team, riducendo il rischio di introduzione di errori. Il Verificatore, a seguito di una revisione positiva, accetta la pull request con la metodologia "squash and merge". Nell'eventualità in cui il branch presenti dei confitti, l'autore si occuperà di risolverli e successivamente dell'unione del branch.

3.3.2.1.1 Elementi Esterni al Repository

Potrebbero esservi delle issue aperte all'interno dell' *ITS*$_G$ che non hanno un corrispondente documento o prodotto in generale, all'interno del repository. Per queste, il ciclo di vita segue il normale flusso attraverso i diversi stati elencati nella Sezione 3.2.2. La verifica viene effettuata attraverso i commenti della issue stessa, che avranno la seguente forma:

- richiesta cambiamenti:

```
[REV]
- richiesta 1;
- richiesta 2;
```

- approvazione:

```
[REV] done
```

3.3.2.2 Analisi statica

L'analisi statica rappresenta un'esplorazione approfondita del codice o della documentazione associata al prodotto. Questa metodologia mira a individuare potenziali problemi o irregolarità, senza mai eseguire effettivamente il sistema software. Nel caso della documentazione, l'analisi statica si concentra sulla struttura, sulla coerenza, sulla completezza e sulla chiarezza del testo. In particolare, verifica la presenza di errori grammaticali, di formattazione e concettuali, garantendo un livello ottimale di qualità nel materiale consegnato.

l verificatore, nel contesto della documentazione o del codice, può condurre l'analisi statica tramite due metodologie: *walkthrough*$_G$ o *inspection*$_G$. Nel caso del nostro progetto, la metodologia prediletta è l' *inspection*$_G$.

3.3.2.2.1 Inspection

In questo processo, il Verificatore adotta un approccio strutturato, seguendo una sequenza di passaggi ben definiti. Utilizza liste di controllo per esaminare

in dettaglio il documento o il codice. Per dettagli specifici sulle checklist usate, si rimanda al documento *PianoDiQualifica_v1.0*.

3.3.2.2.2 Walkthrough

Contrariamente all'*inspection*$_G$, il *walkthrough*$_G$ è un approccio più esplorativo e che lascia maggiore spazio alla collaborazione tra l'autore e il Verificatore. Questa metodologia prevede una lettura a pettine del prodotto con l'obiettivo di analizzare struttura e contenuto nel loro insieme.

La decisione del team di preferire il metodo *inspection*$_G$ è giustificata dall'alto grado di rigore che questo approccio offre e dalla conseguente maggiore efficacia nell' individuazione di tutte le inconsistenze. L'*inspection*$_G$ fornisce una revisione più strutturata e dettagliata, guidata da liste di controllo specifiche, contribuendo a garantire una maggiore completezza e coerenza. Tuttavia, il metodo *walkthrough*$_G$ conserva la sua rilevanza e rimane una valida alternativa di cui ci si può avvalere specialmente nelle fasi iniziali e finali del lavoro su un determinato prodotto: l'approccio pragmatico del metodo risulta infatti adeguato a rilevare criticità e peculiarità che potrebbero non essere rilevate da metodi più formali.

3.3.2.3 Analisi dinamica

L'analisi dinamica si riferisce all'osservazione e alla conseguente valutazione del comportamento di un sistema in esecuzione. Ne risulta che l'oggetto esclusivo di questa metodologia di analisi nel contesto progettuale sia il prodotto risultante dal processo di codifica, il *software*. Nello sviluppo software, l'analisi dinamica è implementata mediante diverse categorie di test. I test, sviluppati a partire da requisiti, funzionali e non, rendono il processo di analisi dinamica ripetibile, e producono un risultato oggettivo. La redazione dei test, e la loro conseguente esecuzione seguono i principi del *Modello a V*$_G$. Il Verificatore si impegna a definire casi di test per ognuna delle seguenti categorie, garantendo così una copertura completa e dettagliata del software:

- Test di unità;
- Test di integrazione;
- Test di sistema;
- Test di accettazione.

La totalità dei test individuati viene riportata all'interno del documento *PianoDiQualifica_v1.0*. In sede di verifica, sulla base del dominio esaminato, il Verificatore è tenuto ad eseguire tali test in maniera rigorosa e a riportarne gli esiti all'interno del *PianoDiQualifica_v1.0*.

3.3.2.4 Classificazione dei test

I test vengono identificati in base alla loro tipologia e tramite un codice numerico. Nello specifico devono avere la seguente forma: **T[Tipologia Test] [Codice]**

Tipologia:
- **U**: unità;
- **I**: integrazione;
- **S**: sistema;
- **A**: accettazione.

Il codice numerico è seriale all'interno della categoria.

3.3.2.5 Stato dei test

Nella sezione relativa ai test nel *Piano di Qualifica* a ogni test viene affiancato il suo stato:
- **N-A** : Il test non è applicabile al contesto attuale o alle funzionalità correnti;
- **N-I** : Il test non è stato implementato;
- **Passato**: Il test ha dato esito positivo;
- **Non Passato**: il test ha dato esito negativo.

3.4 Gestione della Qualità

3.4.1 Descrizione

La gestione della qualità è un insieme di processi e attività volte a garantire che un prodotto soddisfi gli standard di qualità definiti, assicurando che il prodotto sviluppato sia affidabile ed efficiente. È fondamentale garantire che il prodotto soddisfi a pieno i requisiti funzionali e non, stabiliti durante la fase di progettazione.

3.4.2 Obiettivi

- Controllo continuo della qualità del prodotto, verificando che rispetti le aspettative della Proponente;
- Minimizzare la presenza di errori o anomalie nel prodotto;
- Riduzione dei rischi che potrebbero influenzare la qualità;
- Consegna del progetto rispettando il budget preventivato inizialmente e i requisiti individuati assieme alla Proponente.

Per la valutazione della qualità viene fornito il documento *Piano di Qualifica*, in cui sono presenti varie metriche con le relative soglie di valori accettabili ed ideali.

3.4.3 Denominazione Metriche

Per identificare le metriche si usa la seguente formattazione:

M[Tipo]-[Nome]

legenda:

- **Tipo**: specifica la tipologia di metrica:
  1. **PC** se si tratta di qualità di processo;
  2. **PD** se si tratta di qualità di prodotto;
- **Nome**: abbreviazione del nome della metrica.

4 Processi Organizzativi

4.1 Gestione Organizzativa

4.1.1 Decisioni

Per poter prendere una qualsiasi decisione è necessario vi siano due condizioni:

1. Si deve raggiungere il $quorum_G$ di quattro persone su sei;
2. La decisione deve essere verbalizzata e motivata.

4.1.2 Pianificazione

4.1.2.1 Descrizione e scopo

Il Responsabile assume il ruolo cruciale di pianificare dettagliatamente gli obiettivi per ciascuno sprint fino alla conclusione del progetto. Questo implica una distribuzione coerente del lavoro in linea con le scadenze fissate per le revisioni RTB, PB e CA. Il suo compito principale consiste nel delineare chiaramente come il team dovrebbe gestire e completare le attività relative allo sviluppo del software e alla redazione della documentazione in periodi di tempo specifici.

Oltre a definire gli obiettivi per ogni sprint, il Responsabile si occupa di stimare accuratamente il tempo necessario per ciascuna attività e di pianificare la distribuzione dei ruoli all'interno del team. Questa pianificazione è sensata rispetto all'andamento generale del progetto e agli obiettivi imminenti. Queste previsioni vengono formalizzate nel *Piano di Progetto*, diventando il punto di riferimento durante l'evento di Sprint Planning per definire gli obiettivi del successivo sprint.

Eventuali variazioni nella distribuzione dei ruoli, rispetto alla pianificazione iniziale, vengono documentate e giustificate nel **Consuntivo** del *Piano di Progetto*. Questo approccio consente al team di adattarsi in modo flessibile

alle esigenze emergenti, mantenendo costantemente un quadro chiaro delle variazioni e delle ragioni sottendono a tali modifiche.

4.1.2.2 Ruoli Progetto

Durante il periodo di sviluppo del progetto, ciascun membro assume sei ruoli distinti, con compiti diversificati, al fine di garantire una gestione completa ed efficace delle diverse fasi e aspetti del lavoro. Questi ruoli contribuiscono a promuovere la collaborazione sinergica e l'ottimizzazione delle risorse all'interno del team.

I ruoli assunti sono i seguenti:

4.1.2.2.1 Responsabile

Figura chiave che supervisiona, coordina e gestisce le attività del team. Si occupa di garantire l'allineamento tra gli obiettivi del progetto e le azioni intraprese, gestisce le risorse disponibili, stabilisce e mantiene le relazioni esterne con la Proponente e assicura il flusso efficace delle informazioni all'interno del team e al di fuori di esso.

I suoi compiti sono:

- Gestione della comunicazione con la Proponente (si fa uso della piattaforma Element);
- Preparazione dell'ordine del giorno per la successiva riunione, anche sulla base dei punti individuati dagli altri componenti;
- Redazione dei verbali interni ed esterni;
- Stesura e e avanzamento del documento "Piano di progetto";
- Creazione dei diari di bordo;
- Upload dei verbali esterni convalidati, tramite firma, dalla Proponente in una cartella apposita su Google Drive.

4.1.2.2.2 Amministratore

Figura professionale con la responsabilità della creazione, manutenzione e ottimizzazione degli strumenti, delle risorse e dei processi necessari per il corretto svolgimento del progetto.

I suoi compiti sono:

- Configurazione e gestione gli ambienti di sviluppo;
- Implementazione delle procedure operative;
- Assicurare la disponibilità degli strumenti necessari per la collaborazione e la comunicazione all'interno del team;
- Creazione e assegnazione delle *issue* ai membri del team;
- Stesura e avanzamento del documento "Norme di Progetto";

- Implementazione di script dedicati per automatizzare processi nell'ambiente di lavoro;
- Gestione del versionamento dei documenti.

4.1.2.2.3 Analista

Figura professionale che si occupa di analizzare, comprendere e definire i requisiti e le specifiche di un sistema software prima che venga sviluppato. Questa figura svolge un ruolo fondamentale nel processo di sviluppo del software, contribuendo a garantire che il prodotto finale soddisfi le esigenze e le aspettative degli utenti e della Proponente.

I suoi compiti:

- Studio del contesto applicativo e relativa complessità;
- Specifica dei casi d'uso per comprendere in dettaglio i requisiti funzionali del sistema;
- Raccolta dei requisiti per definire le necessità e le funzionalità richieste;
- Stesura del documento *Analisi dei Requisiti*;
- Creazione diagrammi UML;

4.1.2.2.4 Progettista

Figura professionale specializzata nella progettazione architetturale e strutturale di sistemi. La sua responsabilità principale è definire la configurazione, la disposizione e l'organizzazione dei vari componenti del sistema, concentrandosi su come questi elementi interagiscono tra loro per raggiungere determinati obiettivi di funzionalità, prestazioni e scalabilità.

I suoi compiti sono:

- Scelta degli aspetti tecnici e tecnologici;
- Progettazione architetturale che miri all'economicità e alla manutenibilità del sistema;
- Ottimizzazione delle prestazioni usando algoritmi efficienti e gestione memoria;
- Gestione dei rischi: cerca di mitigare problemi che possono sorgere durante lo sviluppo;
- Redazione del documento *Specifiche Tecniche* e di una parte del documento *Piano di Qualifica*.

4.1.2.2.5 Programmatore

Figura professionale incaricata di trasformare le specifiche tecniche in codice eseguibile, garantendo un'implementazione efficiente e accurata delle funzionalità richieste dal progetto.

I suoi compiti:

- Traduzione delle specifiche tecniche in codice funzionante;
- Scrittura di codice chiaro, leggibile e mantenibile;
- Creazione di test per la verifica del software;
- Ampliamento delle *Specifiche Tecniche* conforme alle esigenze del progetto.
- Risoluzione di bug e problemi di performance;
- Realizzazione del *Manuale Utente*;
- Collaborazione con il team per l'integrazione del codice e il mantenimento della coerenza del progetto.

4.1.2.2.6 Verificatore

Figura professionale che si occupa di esaminare il lavoro prodotto dagli altri membri del team.

I suoi compiti:

- Revisione e valutazione della documentazione prodotta dal team;
- Analisi critica del codice per individuare errori, discrepanze o possibili miglioramenti;
- Identificazione e segnalazione di problemi;
- Collaborazione con il team per garantire che il lavoro sia conforme alle linee guida e agli standard richiesti.

4.1.2.3 Cambio dei ruoli

Per garantire che ogni membro svolga almeno una volta tutti i ruoli menzionati, il team si impegna a cambiarli ogni settimana.

4.1.3 Procedure

4.1.3.1 Gestione delle comunicazioni

4.1.3.1.1 Comunicazioni Interne

Riguardano esclusivamente i membri del Team e si svolgono tramite:

- ***Whatsapp***: utilizzato per messaggistica istantanea e una comunicazione veloce;
- ***Discord***: piattaforma utilizzata per:
  1. Creare server suddivisibili in vari canali testuali o vocali, dove verranno svolte le riunioni;
  2. Supplementare la comunicazione all'interno della piattaforma con funzionalita' offerte da servizi esterni quali GitHub.

4.1.3.1.2 Comunicazioni Esterne

Le comunicazioni esterne vengono affidate al Responsabile attraverso i seguenti mezzi:

- *Email* : si usa l'email di gruppo swateng.team@gmail.com;
- *Element*: si usa il canale creato appositamente dalla Proponente per avere una comunicazione diretta.

4.1.3.2 Gestione degli Incontri

4.1.3.2.1 Incontri Interni

Negli incontri interni possono partecipare solamente i membri del gruppo. Si svolgono principalmente una volta a settimana, il giorno può variare in caso di imprevisti, ma solitamente si tiene il venerdì mattina in modalità sincrona.

Le linee guida per le riunioni:

1. Prima dell'incontro avere un ordine del giorno, ovvero i punti eventuali da discutere;
2. Discussione dei punti;
3. Pianificazione attività per la settimana (valutate rispetto a quanto pianificato nel *Piano di Progetto*) e assegnazione issue.

Alla fine dell'incontro:

1. Il Responsabile ha il compito della stesura del verbale interno, fornendo una sintesi dei punti salienti dell'incontro.

Gli incontri hanno due modalità:

- **Fisici**: per gli stand-up meeting quotidiani (*Daily Scrum*$_G$) di 5 minuti in cui si discutono brevemente le attività completate il giorno precedente e si espongono le attività pianificate per il futuro;
- **Virtuali**: si svolgono chiamate o video di gruppo in cui si discutono eventuali dubbi o difficoltà riscontrate. Lo strumento adatto per questo scopo è Discord.

4.1.3.2.2 Incontri Esterni

Negli incontri esterni i partecipanti includono i membri del team e i referenti della Proponente. Questi incontri sono pianificati in concomitanza con l'inizio e la fine dello sprint. Durante queste sessioni, i partecipanti del team hanno l'opportunità di presentare gli sviluppi recenti, condividere i progressi raggiunti e discutere eventuali sfide o questioni emerse nel corso del lavoro. In aggiunta, è possibile richiedere sessioni di formazione mirate su tecnologie specifiche, offrendo al team l'opportunità di approfondire la comprensione di una particolare tecnologia, imparare le *best practice* e acquisire competenze

più avanzate. Il Responsabile ha il compito della stesura del verbale esterno, che viene successivamente convalidato, con firma, dalla Proponente.

4.1.4 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.1.1.

Metrica	Descrizione
BAC	Budget At Completion
EV	Earned Value
PV	Planned Value
SPI	Schedule Performance Index
SV	Schedule Variance

5 Metriche per la qualità

5.1 Metriche per la qualità di processo

5.1.1 Fornitura

- **BAC**: Budget At Completion - indica il budget totale pianificato per il completamento del progetto;
- **EV**: Earned Value - rappresenta il valore prodotto dal progetto ossia il valore dei *deliverable*$_G$ rilasciati fino al momento della misurazione in seguito alle attività svolte

  Formula: $\mathrm{EV} = \text{Percentuale di completamento del lavoro} \cdot \mathrm{BAC}$;
- **PV**: Planned Value - il valore del lavoro pianificato fino a un dato momento

  Formula: $\mathrm{PV} = \text{Percentuale di pianificazione del lavoro} \cdot \mathrm{BAC}$;
- **AC**: Actual Cost - il costo effettivo sostenuto fino a un dato momento;
- **CPI**: Cost Performance Index - misura l'efficienza del costo del lavoro svolto fino a un dato momento

  Formula: $\mathrm{CPI} = \frac{\mathrm{EV}}{\mathrm{AC}}$;
- **SPI**: Schedule Performance Index - misura l'efficienza del tempo rispetto alla pianificazione del progetto. Fornisce un indicatore numerico che rappresenta il rapporto tra il lavoro effettivamente eseguito (o il valore guadagnato) e il lavoro pianificato fino a un determinato punto nel tempo. Aiuta a valutare quanto il progetto sta rispettando il programma pianificato

  Formula: $\mathrm{SPI} = \frac{\mathrm{EV}}{\mathrm{PV}}$;

- **EAC**: Estimated at Completion - revisione del valore stimato per la realizzazione del progetto, ossia il BAC rivisto allo stato corrente del progetto

  Formula: $\mathrm{EAC} = \frac{\mathrm{BAC}}{\mathrm{CPI}}$;
- **ETC**: Estimated To Completion - stima del costo aggiuntivo necessario per completare il progetto

  Formula: $\mathrm{ETC} = \mathrm{EAC} - \mathrm{AC}$;
- **VAC**: Variance at Completion - la differenza tra il budget previsto e il budget attuale alla fine del progetto

  Formula: $\mathrm{VAC} = \mathrm{BAC} - \mathrm{EAC}$;
- **SV**: Schedule Variance - indica se si è in linea, in anticipo o in ritardo rispetto alla schedulazione delle attività di progetto pianificate

  Formula: $\mathrm{SV} = \mathrm{EV} - \mathrm{PV}$;
- **CV**: Cost Variance - la differenza tra il valore del lavoro effettivamente svolto e il costo effettivo del lavoro svolto fino a un dato momento

  Formula: $\mathrm{CV} = \mathrm{EV} - \mathrm{AC}$;
- **BV**: Budget Variance - indica se alla data corrente si è speso di più o di meno rispetto a quanto inizialmente previsto nel budget

  Formula: $\mathrm{BV} = \mathrm{PV} - \mathrm{AC}$.

5.1.2 Codifica

- **CCM**: Complessità Ciclomatica per Metodo - quantifica la complessità del codice misurando il numero di percorsi linearmente indipendenti attraverso il grafo di controllo di flusso del metodo. Più è alta la complessità ciclomatica, maggiore è la complessità del codice

  Formula: $\mathrm{MCCM} = e - n + 2$, con:

  - e: numero di archi del grafo del flusso di esecuzione del metodo;
  - n: numero di vertici del grafo del flusso di esecuzione del metodo.
  - **CC**: Code Coverage - numero di linee di codice convalidate con successo nell'ambito di una procedura di test

    Formula: $\mathrm{MCC} = \frac{\text{linee di codice percorse}}{\text{linee di codice totali}} \cdot 100$;

- **PTCP**: Passed Test Cases Percentage - rappresenta la percentuale di casi di test che sono stati eseguiti con successo rispetto al totale dei casi di test previsti

  Formula: $\mathrm{PTCP} = \frac{\text{test superati}}{\text{test totali}} \cdot 100$;
- **FTCP**: Failed Test Cases Percentage - rappresenta la percentuale di casi di test che non sono stati superati rispetto al totale dei casi di test previsti

  Formula: $\mathrm{FTCP} = \frac{\text{test falliti}}{\text{test totali}} \cdot 100$.

5.1.3 Documentazione

- **IG**: Indice Gulpease - metrica utilizzata per valutare la leggibilità di un testo in lingua italiana. L'Indice Gulpease tiene conto di due variabili linguistiche: la lunghezza delle parole e la lunghezza delle frasi. La formula per calcolare l'indice è la seguente:

  $\mathrm{IG} = 89 + \frac{300 \cdot \mathrm{Nf} - \mathrm{Nl}}{\mathrm{Np}}$, con:

  - **Nf** ⟶ indica il numero delle frasi;
  - **Nl** ⟶ indica il numero delle lettere;
  - **Np** ⟶ indica il numero delle parole.

  L'indice fornisce un punteggio che varia da 0 a 100. Di seguito le possibili interpretazioni:

  - 0-29: Testo difficile da leggere;
  - 30-49: Testo leggibile con sforzo;
  - 50-59: Testo abbastanza leggibile;
  - 60-69: Testo leggibile;
  - 70-79: Testo facile da leggere;
  - 80-89: Testo molto facile da leggere;
  - 90-100: Testo estremamente facile da leggere.
- **CO**: Correttezza ortografica - numero errori grammaticali ed ortografici in un documento.

5.2 Metriche per la qualità di prodotto

- **ROS**: Requisiti Obbligatori Soddisfatti - la percentuale di requisiti obbligatori soddisfatti dal prodotto

  Formula: $\mathrm{ROS} = \frac{\text{requisiti obbligatori soddisfatti}}{\text{requisiti obbligatori totali}} \cdot 100$;
- **RDS**: Requisiti Desiderabili Soddisfatti - la perecentuale di requisiti desiderabili soddisfatti dal prodotto

  Formula: $\mathrm{RDS} = \frac{\text{requisiti desiderabili soddisfatti}}{\text{requisiti desiderabili totali}} \cdot 100$;

- **ROPS**: Requisiti Opzionali Soddisfatti - la percentuale di requisiti opzionali soddisfatti dal prodotto

  Formula: $\mathrm{ROPS} = \frac{\text{requisiti opzionali soddisfatti}}{\text{requisiti opzionali totali}} \cdot 100$;
- **FU**: Facilità di utilizzo - quantità di click che l'utente deve effettuare per raggiungere un obiettivo desiderato;
- **SFIN**: Structure Fan In - rappresenta il numero di moduli o componenti che chiamano un modulo o una funzione specifica; un fan-in elevato indica che molte parti del sistema dipendono da un particolare modulo. Questo può essere un segno positivo di riusabilità, in quanto il modulo è utilizzato in molte parti del sistema;
- **SFOUT**: Structure Fan Out - rappresenta il numero di moduli o funzioni chiamati da un modulo o una funzione specifica; un fan-out elevato può indicare che un modulo ha molte dipendenze da altri moduli. Questo può portare a una maggiore complessità del sistema, poiché le modifiche in un modulo possono richiedere modifiche in molti altri moduli;
- **ATC**: Attributi per Classe - rappresenta il numero di attributi appartenenti ad una classe;
- **PM**: Parametri per Metodo - rappresenta il numero di parametri appartenenti ad un metodo;
- **LCM**: Linee di Codice per Metodo - rappresenta il numero di linee di codice che costituiscono un metodo.