S.W.A.T.
ENGINEERING

Norme di Progetto

Contatti: swateng.team@gmail.com

Versione: 1.1

Registro delle Modifiche

Versione	Data	Descrizione	Autore	Ruolo	Verificatore
1.1	21-02-2024	Ampliata sezione attività "AdR" "Doveri partecipanti" "comandi git" "sigle" e correzioni minori	Riccardo Costantin	Amministratore	Nancy Kalaj, Simone Caregnato
1.0	21-01-2024	Approvazione finale	Nancy Kalaj	Responsabile	
0.18	19-01-2024	Aggiunta sezione "Preventivo"	Riccardo Costantin	Amministratore	
0.17	27-12-2023	Ampliata sezione "Gestione della configurazione" "Gestione organizzativa"	Riccardo Costantin	Amministratore	
0.16	26-12-2023	Aggiunte sezioni "Stile di codifica" "Diagrammi UML delle classi" "Analisi dinamica"	Riccardo Costantin	Amministratore	
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 9
1.1 Scopo del documento 9
1.2 Scopo del prodotto 9
1.3 Glossario 9
1.4 Riferimenti 9
1.4.1 Riferimenti normativi 9
1.4.2 Riferimenti informativi 9
2 Processi primari 10
2.1 Fornitura 10
2.1.1 Descrizione e scopo 10
2.1.2 Rapporti con la Proponente 10
2.1.3 Documentazione fornita 11
2.1.3.1 Analisi dei Requisiti 11
2.1.3.2 Piano di Progetto 11
2.1.3.3 Piano di Qualifica 12
2.1.3.4 Glossario 12
2.1.3.5 Lettera di Presentazione 12
2.1.4 Strumenti 13
2.1.4.1 Google Calendar 13
2.1.4.2 Google Slides 13
2.1.4.3 Google Sheets 13
2.1.4.4 Google Meet 13
2.1.4.5 Online Gantt 13
2.1.4.6 Draw.io 13
2.1.5 Metriche 13
2.2 Sviluppo 14
2.2.1 Descrizione e scopo 14
2.2.2 Attività 14
2.2.2.1 Analisi dei Requisiti 14
2.2.2.1.1 Descrizione 14
2.2.2.1.2 Scopo 14
2.2.2.1.3 Diagrammi UML_G dei casi d'uso 14
2.2.2.1.4 Identificazione dei casi d'uso 17
2.2.2.1.5 Struttura dei casi d'uso 17
2.2.2.1.6 Requisiti 17
2.2.2.1.7 Identificazione dei requisiti 18
2.2.2.1.8 Informazioni aggiuntive 18
2.2.2.1.9 Metriche 19
2.2.2.2 Progettazione 19

2.2.2.2.1 Descrizione e scopo 19
2.2.2.2.2 Diagrammi *UML*$_G$ delle classi 20
2.2.2.2.3 Qualità architettura e progettazione 22
2.2.2.3 Codifica 23
2.2.2.3.1 Descrizione e scopo 23
2.2.2.3.2 Aspettative 23
2.2.2.3.3 Stile di codifica 23
2.2.2.3.3.1 Lunghezza metodi 23
2.2.2.3.3.2 Singola responsabilità 23
2.2.2.3.3.3 Parametri per metodo 24
2.2.2.3.3.4 Univocità dei nomi 24
2.2.2.3.3.5 Type hint 24
2.2.2.3.3.6 Strumenti 24
2.2.2.3.4 Metriche 24
3 Processi di supporto 25
3.1 Documentazione 25
3.1.1 Descrizione e scopo 25
3.1.2 Lista documenti 25
3.1.3 Ciclo di vita dei documenti 25
3.1.4 Template *Typst*$_G$ 26
3.1.5 Nomenclatura 26
3.1.6 Versionamento 26
3.1.7 Struttura 26
3.1.7.1 Prima pagina 26
3.1.7.2 Intestazione 27
3.1.7.3 Registro delle modifiche 27
3.1.7.4 Indice 27
3.1.7.5 Verbali 27
3.1.8 Convenzioni stilistiche 28
3.1.8.1 Elenchi puntati 28
3.1.8.2 Caption 28
3.1.8.3 Formato delle date 28
3.1.8.4 Link 28
3.1.8.5 Sigle 28
3.1.9 Strumenti 29
3.1.10 Metriche 29
3.2 Gestione della configurazione 30
3.2.1 Descrizione e scopo 30
3.2.2 Issue Tracking System 30
3.2.3 Strumento di condivisione 30

3.2.4 Controllo di versione e repository 30
3.2.4.1 Gerarchia file *repository*$_G$ Docs 31
3.2.5 Comandi beginner github 32
3.2.6 Automazioni 32
3.2.6.1 Sito vetrina 32
3.2.6.2 Controllo termini glossario 33
3.3 Verifica 33
3.3.1 Descrizione e scopo 33
3.3.2 Strumenti 33
3.3.2.1 GitHub 33
3.3.2.1.1 Elementi esterni al *repository*$_G$ 34
3.3.3 Analisi statica 34
3.3.3.1 *Inspection*$_G$ 34
3.3.3.2 *Walkthrough*$_G$ 34
3.3.4 Analisi dinamica 34
3.3.4.1 Test di unità 35
3.3.4.2 Test di integrazione 35
3.3.4.3 Test di *sistema*$_G$ 35
3.3.4.4 Test di accettazione 36
3.3.5 Classificazione dei test 36
3.3.6 Stato dei test 36
3.4 Validazione 36
3.4.1 Descrizione e scopo 36
3.4.2 Test di accettazione 36
3.5 Gestione della qualità 37
3.5.1 Descrizione 37
3.5.2 Obiettivi 37
3.5.3 Denominazione metriche 37
3.5.4 Struttura metriche 37
3.5.5 Grafici metriche 37
3.5.6 Metriche 37
4 Processi organizzativi 39
4.1 Gestione organizzativa 39
4.1.1 Decisioni 39
4.1.2 Pianificazione 39
4.1.2.1 Descrizione e scopo 39
4.1.2.2 *Documentazione*$_G$ fine *sprint*$_G$ 39
4.1.2.3 Preventivo 39
4.1.2.4 Consuntivo 40
4.1.3 Ruoli progetto 40

4.1.3.1 Responsabile 40
4.1.3.2 Amministratore 41
4.1.3.3 Analista 41
4.1.3.4 Progettista 41
4.1.3.5 Programmatore 42
4.1.3.6 Verificatore 42
4.1.4 Cambio dei ruoli 42
4.1.5 Tracciamento del tempo speso 42
4.1.6 Gestione strumenti coordinamento 42
4.1.6.1 Ticketing 42
4.1.7 Gestione dei rischi 43
4.1.7.1 Struttura dei rischi 43
4.1.8 Comunicazione e incontri 45
4.1.8.1 Gestione delle comunicazioni 45
4.1.8.1.1 Comunicazioni interne 45
4.1.8.1.2 Comunicazioni esterne 45
4.1.8.2 Gestione degli incontri 45
4.1.8.2.1 Incontri interni 45
4.1.8.2.2 Incontri esterni 45
4.1.8.2.3 Doveri dei partecipanti 46
4.2 Formazione 47
4.2.1 Scopo e aspettative 47
4.2.2 Formazione individuale dei membri 47
4.2.3 Guide e documentazione 47
5 Metriche per la qualità 48
5.1 Metriche per la qualità di processo 48
5.1.1 Fornitura 48
5.1.2 Codifica 48
5.1.3 Documentazione 49
5.1.4 Gestione della qualità 49
5.2 Metriche per la qualità di prodotto 49
5.2.1 Funzionalità 49
5.2.2 Usabilità 49
5.2.3 Affidabilità 50
5.2.4 Manutenibilità 50
5.2.5 Efficienza 50

Elenco delle Figure

Figure 1: Figura rappresentante un attore. 15
Figure 2: Figura rappresentante un caso d'uso. 16
Figure 3: Figura rappresentante una generalizzazione tra attori. 16
Figure 4: Figura rappresentante una generalizzazione tra casi d'uso. 16
Figure 5: Figura rappresentante un' inclusione tra casi d'uso. 16
Figure 6: Figura rappresentante un' estensione tra casi d'uso. 17
Figure 7: Figura rappresentante la relazione di dipendenza. 21
Figure 8: Figura rappresentante la relazione di aggregazione. 21
Figure 9: Figura rappresentante la relazione di composizione. 21
Figure 10: Figura rappresentante la relazione di associazione. 22
Figure 11: Figura rappresentante la relazione di generalizzazione. 22

Elenco delle Tabelle

Table 1: Metriche di fornitura 13
Table 2: Esempio tabella per requisiti 19
Table 3: Metriche sui requisiti 19
Table 4: Metriche di codifica 24
Table 5: Metriche relative alla documentazione 29
Table 6: Metriche relative alla gestione della qualità 38

1 Introduzione

1.1 Scopo del documento

Il presente documento ha l'obiettivo di definire le *best practices*$_G$ e il *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 risposta alle esigenze sviluppatesi e alle problematiche incorse nel corso del lavoro.

1.2 Scopo del prodotto

Il progetto "InnovaCity" si concentra sulla creazione di una *dashboard*$_G$ intuitiva, ovvero facilmente comprensibile e accessibile, che permetta al personale amministrativo di monitorare e analizzare il continuo sviluppo di una *smart city*$_G$. L'applicazione comprende una *data pipeline*$_G$, appositamente 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 v1.0*, 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. L'attività di inserimento di un termine nel glossario può considerarsi completata solo quando il termine viene correttamente definito e spiegato all'interno del *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 (22-02-2024)

1.4.2 Riferimenti informativi

- ***Documentazione*$_G$ git**:
  https://git-scm.com/docs (22-02-2024)
- ***Documentazione*$_G$ GitHub**:
  https://docs.github.com/en (22-02-2024)
- **Materiale didattico del corso Metodi e Tecnologie per lo Sviluppo Software 2022/2023**:
  https://stem.elearning.unipd.it/course/view.php?id=5359 (22-02-2024)
  - Lezione 4: GIT;
  - Laboratorio 2: GitHub Version Control System.
- ***Documentazione*$_G$ *Typst*$_G$**:
  https://typst.app/docs (22-02-2024)

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, si procede con la redazione del *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**:
  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**: individuazione preventiva di una suddivisione precisa di tutte le ore produttive disponibili, seguita da una stima dei costi per ciascun incremento di lavoro (*sprint*$_G$);
- **Esecuzione**: progettazione e sviluppo del prodotto procedono di pari passo con la stesura della *documentazione*$_G$ 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 si è concordato di organizzare incontri regolari, in particolare al termine di ciascuno *sprint*$_G$, fissati per il venerdì alle 10:30. Questi incontri, che assumono la forma di sessioni di *sprint review*$_G$, consentono al team di esporre quanto realizzato e di ricevere feedback sull'andamento del lavoro.

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

- **Incontri di formazione**: finalizzati ad acquisire familiarità con nuove tecnologie, approfondire concetti specifici o migliorare competenze 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*$_G$**: fase conclusiva di ogni *sprint*$_G$ 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 vengono elencati i documenti che il team *SWAT Engineering* si impegna a consegnare ai Committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin, nonché alla Proponente Sync Lab.

2.1.3.1 Analisi dei Requisiti

L'*Analisi dei Requisiti v1.0*, redatto dagli Analisti, rappresenta un documento fondamentale per lo sviluppo del *sistema*$_G$ 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:

- **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*$_G$. I casi d'uso offrono una visione chiara delle azioni eseguibili all'interno del *sistema*$_G$ e delle interazioni degli utenti con esso. All'interno di ciascun caso d'uso, viene fornito un elenco preciso delle azioni intraprese dall'*attore*$_G$ per attivare il caso d'uso, facilitando così l'estrazione dei requisiti corrispondenti;
- **Definizione di requisiti**: individuazione dei requisiti obbligatori, desiderabili e opzionali e loro categorizzazione in:
  - **Requisiti funzionali**: specificano le operazioni che il *sistema*$_G$ deve essere in grado di eseguire;
  - **Requisiti di qualità**: si concentrano sulla definizione degli standard e degli attributi che il software deve possedere per garantire prestazioni, affidabilità, usabilità e sicurezza ottimali;
  - **Requisiti di vincolo**: delineano vincoli e limitazioni che il *sistema*$_G$ deve rispettare. Possono includere restrizioni tecnologiche, normative o di risorse;
  - **Requisiti di prestazioni**: specificano le capacità che il *sistema*$_G$ deve possedere per soddisfare determinate prestazioni, come ad esempio velocità, scalabilità, utilizzo delle risorse, affidabilità, tolleranza agli errori e disponibilità.

2.1.3.2 Piano di Progetto

Il *Piano di Progetto v1.0*, redatto dal Responsabile, 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 possono emergere durante lo sviluppo del progetto.

Si compone delle seguenti sezioni:

- **Analisi dei rischi**: identifica, valuta e gestisce i potenziali rischi che possono 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 e pianificare strategie di mitigazione;
- **Modello di sviluppo**: approccio metodologico 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*$_G$, 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*$_G$, 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*$_G$. 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*$_G$;
- **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*$_G$ e, di conseguenza, anche il costo effettivo. Inoltre, si conduce una breve analisi retrospettiva per giustificare le scelte effettuate nel preventivo, evidenziare eventuali deviazioni e delineare i cambiamenti nella strategia utilizzata per pianificare il resto del lavoro, qualora dovessero rendersi necessari.

2.1.3.3 Piano di Qualifica

Il *Piano di Qualifica v1.0*, redatto dall'Amministratore, 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.

Si compone delle seguenti sezioni:

- **Qualità di processo**: standard e procedure adottate per garantire la qualità durante lo sviluppo del progetto. Include 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;
- **Cruscotto della qualità**: resoconto delle attività di valutazione effettuate durante il progetto. Le valutazioni risultano fondamentali a tracciare l'andamento del progetto rispetto agli obiettivi e alle aspettative, e a identificare prontamente eventuali azioni correttive necessarie a garantire la qualità complessiva del progetto.

2.1.3.4 Glossario

Il *Glossario v1.0* funge da catalogo esaustivo che raccoglie i termini tecnici impiegati all'interno del progetto, offrendo definizioni chiare e precise. Questo documento previene fraintendimenti e promuove una comprensione condivisa della terminologia specifica del settore, migliorando così la coerenza e la qualità della *documentazione*$_G$ prodotta dal team.

2.1.3.5 Lettera di Presentazione

La *Lettera di Presentazione* accompagna la consegna del prodotto software e della *documentazione*$_G$ pertinente, sottolineando 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. In questo contesto, costo e data di consegna del progetto sono attentamente valutati in relazione all'andamento corrente e vengono confermati o eventualmente modificati in risposta alle dinamiche emergenti durante l'evoluzione del progetto.

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*$_G$.

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*$_G$ nella sezione di **Pianificazione** del *Piano di Progetto v1.0*.

2.1.4.6 Draw.io

Software per creare diagrammi e grafici di varia natura, utilizzato dagli Analisti per creare i diagrammi *UML*$_G$ dei casi d'uso nella sezione Casi d'uso dell'*Analisi dei Requisiti v1.0*.

2.1.5 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
SV	Schedule Variance
AC	Actual Cost
CPI	Cost Performance Index
EAC	Estimated At Completion
ETC	Estimated To Completion
VAC	Variance At Completion
BV	Budget Variance

Table 1: Metriche di fornitura

2.2 Sviluppo

2.2.1 Descrizione e scopo

Il processo di sviluppo rappresenta la serie di attività che devono essere svolte affinché il team *SWAT Engineering* 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 Attività

2.2.2.1 Analisi dei Requisiti

2.2.2.1.1 Descrizione

L'*Analisi dei Requisiti v1.0* 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;
- **Attori**: gli utilizzatori del prodotto finale;
- **Casi d'uso**: individua gli attori e tutte le interazioni che possono avere con il *sistema*$_G$;
- **Requisiti**: le caratteristiche da soddisfare e le fonti da cui sono state estratte.

2.2.2.1.2 Scopo

Lo scopo dell'*Analisi dei Requisiti v1.0* è definire in modo dettagliato e chiaro le funzionalità e le caratteristiche che il prodotto software deve soddisfare. Questo processo mira a comprendere a fondo le esigenze degli utenti, gli obiettivi del *sistema*$_G$ e le condizioni in cui dovrà operare. Gli obiettivi principali dell'attività di analisi dei requisiti includono:

- Identificare e chiarire gli obiettivi e le finalità del prodotto che si intende sviluppare;
- Fornire ai Progettisti una base chiara e comprensibile per la definizione dell'*architettura*$_G$ e il design del *sistema*$_G$;
- Fornire una base per la pianificazione mediante i requisiti raccolti;
- Facilitare la comunicazione tra fornitori e Proponente;
- Fornire riferimenti per la verifica.

2.2.2.1.3 Diagrammi *UML*$_G$ 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*$_G$. La sua utilità risiede nel tracciare i flussi operativi attraverso una rappresentazione visiva, descrivendo il modo in cui un utente interagisce con il *sistema*$_G$.

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*$_G$.

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*$_G$.

I diagrammi dei casi d'uso sono composti da:

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

  A livello di diagramma, l'*attore*$_G$ è simboleggiato da un'icona umana stilizzata, identificabile mediante un'etichetta univoca e rappresentativa, posizionata sotto di essa.

Figure 1: Figura rappresentante un attore.

- **Caso d'uso**: delinea le operazioni eseguibili dall'utente sul *sistema*$_G$. Un singolo caso d'uso si compone di una breve esposizione delle funzioni messe a disposizione del *sistema*$_G$ 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 si sta trattando un eventuale sottocaso 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*$_G$ principale**: l'*attore*$_G$ che intende compiere lo scopo rappresentato dal caso d'uso;
  - **Precondizioni**: stato in cui il *sistema*$_G$ si deve trovare prima dell'avvio della funzionalità rappresentata dal caso d'uso;
  - **Postcondizioni**: stato in cui il *sistema*$_G$ si troverà dopo che l'utente avrà portato a termine lo scopo rappresentato dal caso d'uso;
  - **Scenario principale**: descrizione accurata della funzionalità rappresentata dal caso d'uso;
  - **Specializzazioni**: nel caso di un caso d'uso 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.

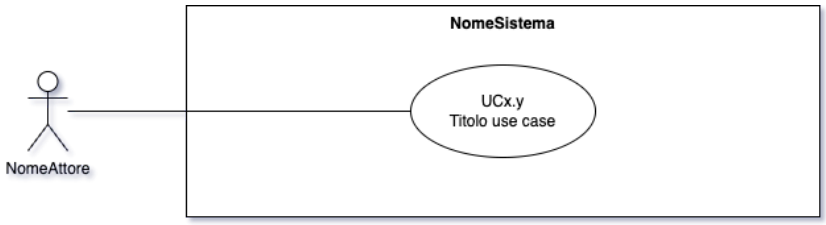

Figure 2: Figura rappresentante un caso d'uso.

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*$_G$ si verifica quando un *attore*$_G$ di livello superiore, dotato di abilità più generiche, viene specializzato in comportamenti più specifici nei sottostanti attori. Ogni *attore*$_G$ sottostante eredita le funzionalità dal suo *attore*$_G$ 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.

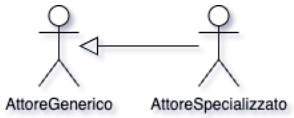

Figure 3: Figura rappresentante una generalizzazione tra attori.

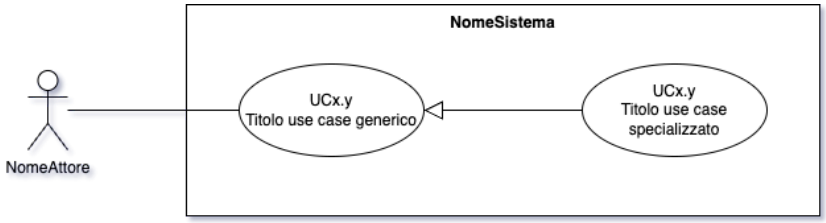

Figure 4: Figura rappresentante una generalizzazione tra casi d'uso.

- **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".

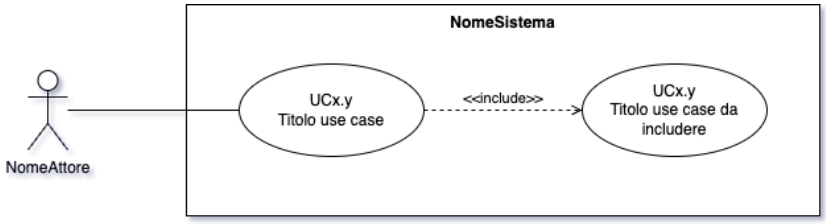

Figure 5: Figura rappresentante un' inclusione tra casi d'uso.

- **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".

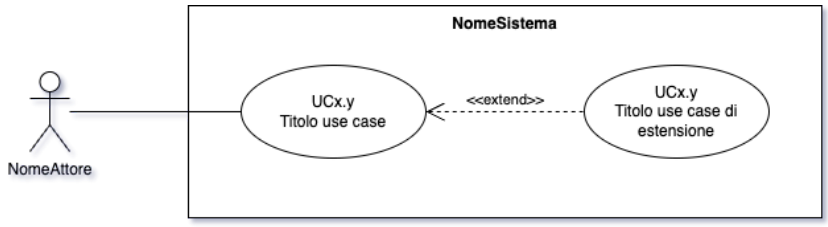

Figure 6: Figura rappresentante un' estensione tra casi d'uso.

2.2.2.1.4 Identificazione dei casi d'uso

Un caso d'uso rappresenta un singolo scenario o interazione tra un *attore*$_G$ e il *sistema*$_G$ software, focalizzandosi sulle azioni compiute dall'*attore*$_G$ e sulle risposte del *sistema*$_G$ a tali azioni. 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.

2.2.2.1.5 Struttura dei casi d'uso

I casi d'uso sono strutturati nel seguente modo:

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

2.2.2.1.6 Requisiti

I requisiti trovati vengono classificati nei seguenti modi:

- **Funzionali**:

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

- **Qualità**:

  un requisito di qualità stabilisce gli standard e i criteri che il *sistema*$_G$ 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;

- **Prestazionale**:

  un requisito di prestazione è una specifica che descrive le prestazioni o le capacità che il *sistema*$_G$ deve soddisfare.

2.2.2.1.7 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;
  - P: se di prestazioni.
- **Numero**: per ogni requisito aggiunto il numero viene incrementato.

2.2.2.1.8 Informazioni aggiuntive

All'interno del documento *Analisi dei Requisiti v1.0*, è importante organizzare le informazioni in modo strutturato per garantire una comprensione chiara e completa di ciascun requisito. Viene dunque utilizzata una tabella per specificare le informazioni aggiuntive pertinenti ai requisiti di ciascun tipo. Le tabelle vengono strutturate nel seguente modo:

- **Codice**: il codice del requisito;
- **Importanza**: l'importanza del requisito (obbligatorio, opzionale, desiderabile);
- **Descrizione**: breve descrizione per ridurre le ambiguità del requisito e assicurarsi che sia atomico nella sua definizione;
- **Fonte**: viene specificata la fonte da cui il requisito è stato tratto, il che risulta particolarmente utile per facilitare il tracciamento dei requisiti. Possibili fonti includono il capitolato, un verbale esterno, le *Norme di Progetto* o i vari casi d'uso.

Per costruire le tabelle è stata usata un'automazione in *Typst*$_G$, che prende in input semplicemente l'importanza, la descrizione e la fonte del singolo requisito, utilizza una funzione apposita per generare il codice univoco del requisito e costruisce la tabella contenente tutti requisiti di una particolare categoria. Esempio:

```
#let requisiti_vincolo = (
    (
        "Obbligatorio","La creazione di un simulatore di almeno una sorgente
dati.","Capitolato",
    )
)

#let requisiti_vincolo_con_codice = generate_requirements_array("V",
```

```
requisiti_vincolo)
#figure(
requirements_table(requisiti_vincolo_con_codice)
```

“generate_requirement_array” è la funzione per generare il codice del requisito mentre “requirements_table” è la funzione per generare la tabella.

Codice	Importanza	Descrizione	Fonti
ROV1	Obbligatorio	La creazione di un simulatore di almeno una sorgente dati.	Capitolato

Table 2: Esempio tabella per requisiti

2.2.2.1.9 Metriche

La definizione delle metriche seguenti si può trovare nella Sezione 5.2.1.

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

Table 3: Metriche sui requisiti

2.2.2.2 Progettazione

2.2.2.2.1 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 v1.0*. 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*$_G$ in componenti specifiche e definendo le interazioni tra di esse. 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. **Requirements and Technology Baseline (*RTB*$_G$)**: implica la selezione e la definizione delle tecnologie di base utilizzate per la realizzazione del *sistema*$_G$. Questo comprende decisioni relative a linguaggi di programmazione, librerie e *framework*$_G$. Tale processo porta alla creazione di un Proof of Concept (*PoC*$_G$);

   Include:
   - **Proof of Concept (*PoC*$_G$)**: 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*$_G$, anche su consiglio della Proponente.
2. **Progettazione architetturale**: definizione ad alto livello dell’*architettura*$_G$ del *sistema*$_G$; si concentra sulla suddivisione del *sistema*$_G$ 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 (*PB$_G$*)**: segna un punto stabile nell'attività di progettazione, in cui le specifiche tecniche, le funzionalità principali e l'*architettura$_G$* 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$_G$* 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$_G$*.
   - Test di unità su ogni componente.

2.2.2.2.2 Diagrammi *UML$_G$* delle classi

Il diagramma delle classi è una tipologia di diagramma *UML$_G$* utile a rappresentare la struttura statica di un *sistema$_G$* orientato agli oggetti. Esso visualizza le classi del *sistema$_G$*, insieme ai loro attributi e metodi, e le relazioni tra di esse. Le classi sono rappresentate tramite rettangoli divisi in tre sezioni: la parte superiore contiene il nome della classe, la sezione centrale include gli attributi della classe e quella inferiore ne elenca i metodi.

- **Nome**: nome della classe in grassetto, se la classe è astratta viene scritto in corsivo oltre che in grassetto;
- **Attributi**:

  Visibilità Nome : **Tipo** [Molteplicità] = {Default}

  Legenda:
  - **Visibilità**: può essere di 3 tipi;
    1. Privata, viene indicata con il -;
    2. Protetta, viene indicata con il #;
    3. Pubblica, viene indicata con il +.
  - **Nome**: il nome dell'attributo, se statico viene sottolineato;
  - **Tipo**: rappresenta il tipo di dato dell'elemento;
  - **Molteplicità**: quante istanze dell'elemento possono esistere in relazione ad altri elementi;
  - **Default**: se configurato, indica il valore predefinito per l'elemento.
- **Metodi**:

  Visibilità Nome (Lista-Parametri) : **Ritorno**

  Legenda:
  - **Visibilità**: segue quanto sopra;
  - **Nome**: nome del metodo, se statico viene sottolineato;
  - **Lista-Parametri**: se la funzione prevede più di un parametro, questi vengono separati tramite ",";
  - **Ritorno**: il tipo restituito dal metodo.

Di seguito si elencano le possibili relazioni:

- **Dipendenza**: una dipendenza tra due classi è indicata da una freccia tratteggiata con la punta, che parte dalla classe dipendente A e punta alla classe da cui si origina la dipendenza B. Questa freccia simbolizza il fatto che un cambiamento nella classe B può avere un impatto o influenzare la classe A;

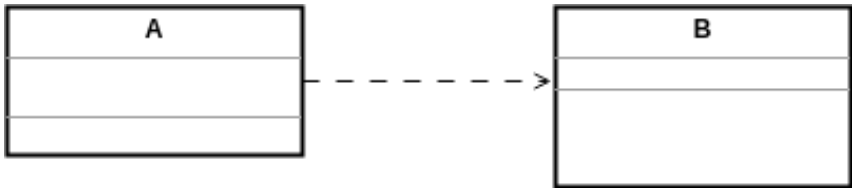

Figure 7: Figura rappresentante la relazione di dipendenza.

- **Aggregazione**: una classe A contiene un riferimento a un oggetto di tipo B, condividendo questo riferimento con altre classi. In sostanza, la classe A possiede un campo di tipo B che viene assegnato da una fonte esterna. L'oggetto di tipo B può essere utilizzato anche da altre parti del *sistema*$_G$, non solo dalla classe A. Questa relazione è indicata da una freccia a rombo vuoto, in cui A rappresenta l'oggetto contenitore e B il contenuto;

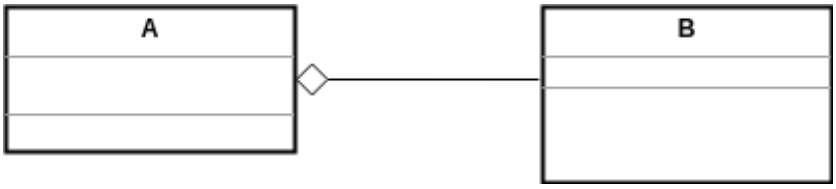

Figure 8: Figura rappresentante la relazione di aggregazione.

- **Composizione**: una classe A contiene un oggetto di tipo B. In questa relazione, la classe A ha la responsabilità di creare e gestire l'oggetto di tipo B. È una relazione "parte-tutto", in cui l'oggetto di tipo B esiste solo all'interno dell'oggetto di tipo A e dipende strettamente da esso. Se la classe A viene eliminata, anche la classe B associata viene cancellata. Questa relazione è indicata da una freccia a rombo pieno, dove A rappresenta l'oggetto contenitore e B il contenuto;

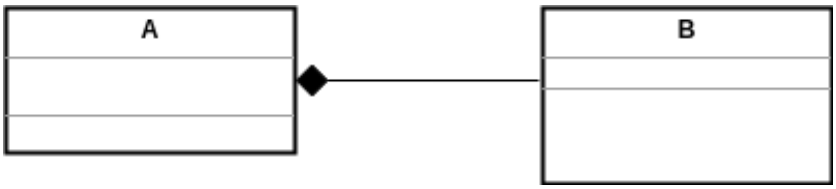

Figure 9: Figura rappresentante la relazione di composizione.

- **Associazione**: una classe A contiene riferimenti o istanze di un'altra classe B. Questa relazione non implica una dipendenza stretta o un'inclusione di un'istanza all'interno dell'altra.

La rappresentazione grafica di un'associazione consiste in una linea tra le classi coinvolte, spesso accompagnata da molteplicità che indica quanti oggetti di una classe sono associati a quanti oggetti dell'altra classe. Questi valori di molteplicità possono essere situati agli estremi della linea che collega le classi.

Viene rappresentata con una freccia;

Figure 10: Figura rappresentante la relazione di associazione.

- **Generalizzazione**: una classe B eredita attributi, comportamenti e relazioni dalla classe genitore A. In altre parole, ogni istanza di B è anche un'istanza di A (subtyping), ma con specificità o dettagli aggiuntivi propri di B.

Nella rappresentazione grafica, si utilizza una freccia vuota che punta dalla classe figlia (Classe B) alla classe genitore (Classe A).

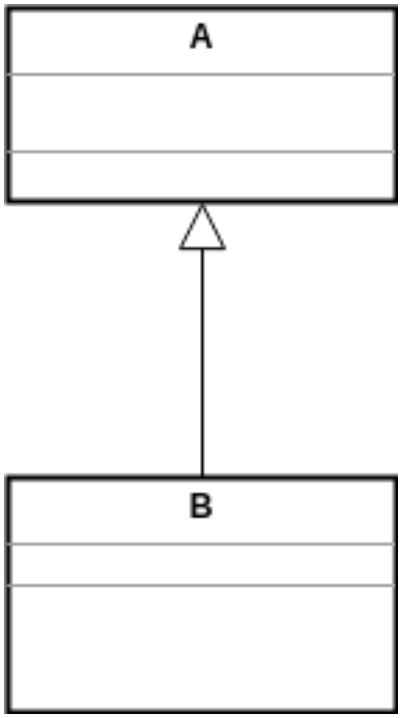

Figure 11: Figura rappresentante la relazione di generalizzazione.

2.2.2.2.3 Qualità architettura e progettazione

La definizione dell'architettura del prodotto è affidata ai Progettisti. Questa serve da base per lo sviluppo del software e deve garantire che le varie componenti siano facilmente identificabili, coese e riutilizzabili. Per garantire la qualità dell'architettura, i Progettisti possono fare affidamento su pattern architetturali consolidati e scelti opportunamente, vari tool e *framework*$_G$ e ottime pratiche di documentazione. I Progettisti, infine, devono garantire che la progettazione del software sia di alta qualità, per garantire funzionalità, prestazioni, affidabilità, sicurezza, manutenibilità e usabilità. L'architettura deve soddisfare vari criteri:

- Soddisfazione dei requisiti elencati nel documento *Analisi dei Requisiti v2.0*;
- Tracciamento dei requisiti: deve essere presente un tracciamento completo dei requisiti attraverso l'implementazione di una tabella che comprende il codice del requisito, la sua descrizione e un indicazione se è stato soddisfatto o meno. In aggiunta, è richiesto un grafico a torta che rappresenti la percentuale di requisiti soddisfatti rispetto al totale;
- Utilizzo efficace ed efficiente delle risorse;

- Adozione di ottime pratiche di documentazione, che vengono rispecchiate all'interno delle *Specifiche Tecniche*.

2.2.2.3 Codifica

2.2.2.3.1 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 operano all'interno di un contesto ben definito seguendo le linee guida stabilite durante la fase di progettazione architetturale. Questo approccio garantisce coerenza nell'implementazione del design e nell'applicazione delle *best practices*$_G$, favorendo la creazione di codice robusto, manutenibile e di alta qualità.

Nel perseguire gli obiettivi di qualità, i Programmatori sono tenuti a rispettare le metriche definite nel *Piano di Qualifica v1.0*.

2.2.2.3.2 Aspettative

La fase di codifica è intrinsecamente orientata alla creazione di un prodotto software che soddisfi pienamente le esigenze e le aspettative della Proponente. 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.

2.2.2.3.3 Stile di codifica

Il team *SWAT Engineering* ha deciso di adottare il linguaggio di programmazione *Python*$_G$ per la creazione dei simulatori di sensori. Questa scelta tecnologica è motivata dalla familiarità che molti membri del team hanno con il linguaggio in questione, il che consente al team di concentrarsi direttamente sulla realizzazione del prodotto, risparmiando una quantità significativa risorse. Inoltre, *Python*$_G$ promuove uno stile di codifica uniforme e leggibile. La sua struttura rigida basata sull'indentazione agevola la comprensione della struttura del programma, contribuendo a una maggiore chiarezza all'interno del codice. Il team si impegna ad utilizzare i seguenti stili di codifica:

2.2.2.3.3.1 Lunghezza metodi

Sono da preferire metodi di lunghezza limitata per diverse ragioni tra cui:
- Leggibilità;
- Manutenibilità;
- Riusabilità;
- Testabilità;
- Analisi statica semplificata.

2.2.2.3.3.2 Singola responsabilità

Ogni classe o funzione deve avere un unico obiettivo ben definito. Quando ogni entità nel codice svolge un compito specifico, ciò garantisce che ci sia un motivo chiaro e singolare per apportare modifiche. Questo approccio non solo promuove la chiarezza e la manutenibilità del codice, ma contribuisce anche a ridurre la complessità ciclomatica del software.

2.2.2.3.3.3 Parametri per metodo

Preferire l'uso di metodi con un numero limitato di parametri quando possibile. Questo approccio offre diversi vantaggi che contribuiscono alla chiarezza, manutenibilità e testabilità del codice.

2.2.2.3.3.4 Univocità dei nomi

Tutte le variabili, i metodi e le classi devono avere un nome che li distingue univocamente, per limitare la possibilità di ambiguità del codice.

2.2.2.3.3.5 Type hint

L'utilizzo dei type hint, ossia l'annotazione dei tipi di dati nelle firme delle funzioni o dei metodi, pratica comunemente utilizzata in *Python*$_G$, contribuisce ad una migliore robustezza del codice fornendo alcuni vantaggi:

- Analisi statica avanzata;
- Riduzione errori;
- Contratto esplicito tra chiamante e funzione;
- Facilità di manutenzione.

I type hint non impongono un ritorno specifico, ma forniscono informazioni chiare sui tipi di dati attesi nei parametri e nei risultati delle funzioni.

2.2.2.3.3.6 Strumenti

Si è deciso di integrare il linter PEP8 come fondamentale strumento nel processo di sviluppo. Questo strumento svolge un ruolo di rilievo nel garantire la coerenza e l'adesione alle linee guida di formattazione del codice *Python*$_G$, stabilite nella PEP 8, contribuendo in modo significativo a mantenere uno standard uniforme nella base di codice.

2.2.2.3.4 Metriche

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

Metrica	Descrizione
ATC	Attributi per Classe
PM	Parametri per Metodo
LCM	Linee di Codice per Metodo

Table 4: Metriche di codifica

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione e scopo

La *documentazione*$_G$ 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, garantendone la coerenza mediante il tracciamento di tutti i processi e le attività coinvolte. Ciò mira a 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 v2.0*;
- *Piano di Progetto v2.0*;
- *Piano di Qualifica v2.0*;
- *Analisi dei Requisiti v2.0*;
- *Specifiche Tecniche v1.0*;
- *Manuale Utente v1.0*;
- *Glossario v2.0*;
- *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*$_G$ Docs e si mette in uso;
2. Si copia dall'apposita *repository*$_G$ `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>,<verificatore>`; 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*$_G$ 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 il branch creato, solo quando la pull request viene chiusa o risolta.

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.

Nel contesto della versione finale di un documento, spetta al Responsabile conferire l'approvazione definitiva, annotando opportunamente nel changelog la versione **x.0** e la sua approvazione finale.

In cui **x** può essere:
- 1 se per *RTB*$_G$;
- 2 se per *PB*$_G$;
- 3 se per *CA*$_G$.

3.1.4 Template *Typst*$_G$

Per la stesura dei documenti viene usato un template in formato *Typst*$_G$. 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*$_G$:
- *documentazione*$_G$ 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 nomenclatura 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 `aammdd` 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 si effettua 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 3.1.8.3;
- **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

Il registro delle modifiche è un elenco dettagliato che tiene traccia di tutte le modifiche apportate al documento nel corso del tempo. È utile per tenere traccia dell'evoluzione del documento e per consentire a chiunque stia lavorando sul progetto di comprendere quali modifiche sono state apportate e quando.

Si compone di una 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;
- **Verificatore**: il membro che si è occupato della revisione di quella modifica.

Per quanto riguarda l'ultima colonna, è stata aggiunta solamente in seguito al feedback ricevuto durante la revisione RTB_G; pertanto, le modifiche effettuate precedentemente l'RTB_G, pur essendo state verificate, non riportano esplicitamente il membro che ha effettuato la revisione all'interno del registro.

3.1.7.4 Indice

Nella pagina successiva al registro delle modifiche 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.

Per indicare i partecipanti presenti all'incontro si utilizza un file csv, dove in ogni riga si ha il partecipante e la durata della sua presenza separati da virgola; questo file viene poi convertito automaticamente in una tabella da un'apposita funzione $Typst_G$. Si utilizza inoltre il file `meta.typ` per indicare l'ora di inizio e fine incontro e luogo in cui si è svolto, è sufficiente modificarlo con i dati opportuni in modo che vengano automaticamente riportati nel documento.

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, seguiti dalla loro versione (1.0 nel caso di documenti pronti ad essere presentati alla revisione *RTB*$_G$);
  - I termini di glossario (seguiti da $_G$).
- **Caratteri maiuscoli**:
  - Le iniziali dei nomi;
  - Le lettere che compongono un acronimo e le iniziali della rispettiva definizione;
  - Le iniziali dei ruoli svolti dai componenti del gruppo;
  - L'iniziale del termine "Proponente";
  - Prima lettera di ogni elenco puntato.
- **Monospace**:
  - Per indicare parti di codice in *Typst*$_G$ o in altri linguaggi.

Nei verbali interni ed esterni non si usa la formattazione da glossario.

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 Caption

Ogni immagine o tabella deve avere obbligatoriamente una caption associata, utile a fornire una breve descrizione o spiegazione del contenuto visivo.

3.1.8.3 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.

3.1.8.4 Link

Per i link si usa la seguente formattazione e vanno sottolineati attraverso `\#show link: underline`:

```
#link("https://example.com")
```

Inoltre tra parentesi "()" deve essere presente la data di ultimo accesso al link.

3.1.8.5 Sigle

Le varie sigle relative ai documenti e al progetto sono le seguenti:

- **Documentazione**:
  - Analisi dei Requisiti → **AdR**;
  - Norme di Progetto → **NdP**;
  - Piano di Progetto → **PdP**;

  - Piano di Qualifica → **PdQ**;
  - Manuale Utente → **MU**;
  - Specifiche Tecniche → **ST**;
  - Verbale Interno → **VI**;
  - Verbale Esterno → **VE**;
  - Glossario → **Gls**.
- **Progetto**:
  - Requirements and Technology Baseline → **RTB**;
  - Product Baseline → **PB**;
  - Customer Acceptance → **CA**;
  - Proof of Concept → **PoC**.
- **Ruoli**:
  - Responsabile → **Re**;
  - Amministratore → **Am**;
  - Analista → **An**;
  - Progettista → **Pt**;
  - Programmatore → **Pr**;
  - Verificatore → **Ve**.

3.1.9 Strumenti

Il gruppo utilizza:

- ***Typst***$_G$: 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*$_G$ 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

Table 5: Metriche relative alla documentazione

3.2 Gestione della configurazione

3.2.1 Descrizione e scopo

La gestione della configurazione (configuration management) è una pratica fondamentale nell'ambito dello sviluppo software, specialmente quando si tratta di gestire e controllare configurazioni complesse di prodotti. Essa riguarda il processo di identificazione e controllo delle componenti di un *sistema*$_G$, così come delle relazioni tra di esse, durante tutto il ciclo di vita del prodotto. Consente di mantenere la tracciabilità, la coerenza e la comprensione chiara delle relazioni tra le varie parti del *sistema*$_G$.

3.2.2 Issue Tracking System

Come *ITS*$_G$ si utilizza Github che, attraverso le funzioni di "Project", "Issue" e "Pull request", garantisce una struttura all'organizzazione di progetto.

Le *corsie di Stato*$_G$ descrivono lo stato attuale delle attività, all'interno del Project nell'*ITS*$_G$ 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*$_G$ (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 Controllo di versione e repository

Come sistema di controllo di versione si utilizza Git.

Vengono utilizzate le seguenti *repository*$_G$:

- https://github.com/SWATEngineering/Docs: questa *repository*$_G$ viene impiegata dal team per condividere e revisionare il codice sorgente legato alla *documentazione*$_G$ 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*$_G$;
- https://github.com/SWATEngineering/SWATEngineering.github.io: questa *repository*$_G$ funge da piattaforma per i documenti compilati e approvati. Il sito web https://swatengineering.github.io/ rappresenta la versione web della *documentazione*$_G$ 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*$_G$ 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*$_G$ 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*$_G$ 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 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*$_G$ 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.

3.2.4.1 Gerarchia file *repository*$_G$ Docs

All'interno della cartella `src`, sono presenti 2 cartelle organizzate nel modo seguente (non sono presenti i file compilati in formato `.pdf` poiché un'automazione Github Action si occupa di effettuare la compilazione automatica dei documenti e renderli disponibili all'interno di un artefatto):

- **2_RTB**:
  - AnalisiDeiRequisiti;
  - Glossario;
  - LetteraDiPresentazioneRTB;
  - NormeDiProgetto;
  - PianoDiProgetto;
  - PianoDiQualifica;
  - VerbaliEsterni;
  - VerbaliInterni.
- **3_PB**:
  - AnalisiDeiRequisiti;
  - Glossario;
  - LetteraDiPresentazionePB;
  - ManualeUtente;
  - NormeDiProgetto;
  - PianoDiProgetto;
  - PianoDiQualifica;
  - SpecificheTecniche;
  - VerbaliEsterni;
  - VerbaliInterni.
- **4_CA**:
  - AnalisiDeiRequisiti;
  - Glossario;
  - LetteraDiPresentazioneCA;
  - ManualeUtente;
  - NormeDiProgetto;
  - PianoDiProgetto;

- PianoDiQualifica;
- SpecificheTecniche;
- VerbaliEsterni;
- VerbaliInterni.

3.2.5 Comandi beginner github

Breve elenco dei comandi Git che i membri del team *SWAT Engineering* utilizzano durante il progetto:

- `git clone URL_repo`: per clonare il *repository*$_G$ git remoto in locale. Si preferisce la clonazione tramite SSH;
- `git add .`: per aggiungere tutti i file modificati all'area di staging; in caso non si voglia aggiungerli tutti, occorre specificare il nome di ciascun file da aggiungere singolarmente, assicurandosi anche di essere nella cartella corretta;
- `git commit -m "nome_messaggio"`: per registrare i cambiamenti apportati ad un determinato prodotto, documentale o software; i commit possono essere visti come istantanee dello stato di un prodotto in un dato momento. `-m` specifica il messaggio del commit, e deve essere una descrizione breve ma quanto più dettagliata dei cambiamenti apportati;
- `git push`: le modifiche locali registrate vengono inviate al branch remoto;
- `git pull`: per importare le modifiche effettuate su un branch remoto all'interno del branch locale corrispondente;
- `git pull --rebase origin nome_branch`: in caso più persone stiano lavorando allo stesso branch e si verifichino conflict merge al momento del `git pull`, fa il rebase del branch locale corrente per ottenere la linea cronologica dei vari commit eseguiti e aggiornarlo rispetto alla *repository*$_G$ remota;
- `git fetch origin`: per recuperare tutti i riferimenti remoti;
- `git rebase origin/main`: per fare il rebase in caso il main riceva aggiornamenti importanti, in modo che vengano importati all'interno del branch di lavoro corrente;
- `git branch nomenuovobranch`: per creare nuovo branch;
- `git checkout nomebranch`: per spostarsi tra i vari branch;
- `git stash`: per scartare le modifiche in area di staging.

Per effettuare il merge nel main si utilizza l'interfaccia di Github.

3.2.6 Automazioni

3.2.6.1 Sito vetrina

Il sito vetrina consultabile al seguente link https://swatengineering.github.io/ è progettato per garantire un'esperienza aggiornata e intuitiva agli utenti. Grazie all'integrazione di una GitHub Action, il sito si aggiorna automaticamente ogni volta che vengono apportate modifiche al branch principale "main" del *repository*$_G$ Docs. Lo script, realizzato in JavaScript, avvia una richiesta GET a un server esterno per ottenere la struttura aggiornata delle directory e dei file. Questa operazione consente al sito di rimanere sempre allineato con le modifiche apportate al *repository*$_G$ principale. Per garantire la coerenza e la tempestività delle informazioni, il servizio esterno esegue un aggiornamento periodico, ogni 15 minuti. Questo processo automatico consente al sito di riflettere dinamicamente qualsiasi modifica rilevante nel *repository*$_G$ Docs, garantendo un'esperienza utente sempre aggiornata e completa.

3.2.6.2 Controllo termini glossario

È stato sviluppato uno script in *Python*$_G$ per verificare la corrispondenza tra i termini presenti in un documento e quelli presenti nel *Glossario v1.0*. Lo script, utilizzando espressioni regolari, confronta i termini indicati nei documenti, formattati secondo lo standard del glossario, con quelli presenti nel *Glossario*. In presenza di discrepanze, genera un messaggio di errore specificando i termini mancanti nel *Glossario*. In caso contrario, l'esecuzione avviene senza problemi, indicando che i termini nel documento sono in linea con quelli del *Glossario v1.0*. Tutti i termini inclusi nel *Glossario v1.0* devono essere formattati secondo la convenzione da glossario ogni volta che compaiono nel documento, non solo alla prima occorrenza. Questo assicura coerenza e facilita la consultazione, contribuendo a una migliore comprensione dei termini chiave nel contesto del documento.

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 v1.0*. Il *Piano di Qualifica v1.0* funge da punto di riferimento per il Verificatore: in esso vengono fornite tutte le linee guida a cui il Verificatore deve aderire, garantendo uniformità, coerenza e ripetibilità al processo di verifica.

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 nuovamente la revisione. Il processo viene ripetuto fino a che la revisione non da esito positivo. A seguito del processo di verifica il Verificatore si occupa di spostare la issue nella *corsia*$_G$ 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 nel ramo principale, sia stato effettivamente revisionato da almeno un membro del team, riducendo il rischio di introdurre 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 conflitti, è responsabilità dell'autore risolverli prima di procedere con l'unione del branch. Una volta effettuato il merge, il Verificatore si occupa dell'eliminazione sicura del branch.

3.3.2.1.1 Elementi esterni al *repository*$_G$

Potrebbero esservi delle issue aperte all'interno dell'*ITS*$_G$ che non hanno un corrispondente documento o prodotto in generale, all'interno del *repository*$_G$. 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
```

3.3.3 Analisi statica

L'analisi statica rappresenta un'esplorazione approfondita del codice o della *documentazione*$_G$ associata al prodotto. Questa metodologia mira a individuare potenziali problemi o irregolarità, senza mai eseguire effettivamente il *sistema*$_G$ software. Nel caso della *documentazione*$_G$, 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.

Il Verificatore, nel contesto della *documentazione*$_G$ o del codice, può condurre l'analisi statica tramite due metodologie: *walkthrough*$_G$ o *inspection*$_G$. La metodologia preferita dal team, e utilizzata con maggior frequenza, è l'*inspection*$_G$.

3.3.3.1 *Inspection*$_G$

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 *Piano di Qualifica v1.0*.

3.3.3.2 *Walkthrough*$_G$

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.

3.3.4 Analisi dinamica

L'analisi dinamica nel contesto dello sviluppo software si concentra sull'osservazione e valutazione del comportamento del *sistema*$_G$ durante l'esecuzione. Questa metodologia è specificamente rivolta al prodotto software risultante dall'attività di codifica. Nel processo di

sviluppo, l'analisi dinamica è attuata attraverso varie categorie di test. I test, derivati dai requisiti, siano essi funzionali o non funzionali, rendono l'attività di analisi dinamica ripetibile. Questo significa che è possibile eseguire i test più volte con gli stessi input e condizioni, ottenendo risultati coerenti e affidabili. La ripetibilità dei test è fondamentale per confermare la coerenza delle funzionalità del software in diverse situazioni e sotto diverse condizioni operative. La definizione e l'esecuzione dei test seguono i principi del *Modello a V_G*.

Il Verificatore si impegna a elaborare casi di test per ciascuna delle seguenti categorie, garantendo 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 *Piano di Qualifica 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 *Piano di Qualifica v1.0*.

3.3.4.1 Test di unità

Il test di unità è una fase del processo di software testing che si concentra sulla verifica delle singole unità o componenti del software. Un'unità è la più piccola parte di un'applicazione che può essere testata isolatamente. Gli obiettivi principali:

- Testare ogni unità in modo separato, isolando le influenze esterne e garantendo che ciascuna parte del software funzioni correttamente da sola;
- Accertarsi che ciascuna unità esegua le operazioni specificate nel suo design;
- Identificare precocemente i possibili errori.

I test di unità si dividono principalmente in due categorie:

- **Test Funzionali**: verificano che ciascuna unità esegua le funzioni specificate nel design, concentrandosi sulla logica interna dell'unità e testando i casi in cui la funzione produce i risultati desiderati;
- **Test Strutturali**: verificano la struttura interna dell'unità, compresa la logica di controllo e il flusso dei dati, esaminando il codice sorgente dell'unità.

3.3.4.2 Test di integrazione

I test di integrazione sono cruciali per valutare il comportamento delle unità software quando vengono combinate. Una volta confermato il corretto funzionamento delle singole unità in isolamento, i test di integrazione mirano a identificare possibili problemi nelle interazioni tra le unità integrate. Questi test, automatizzati il più possibile, verificano se le componenti del software collaborano efficacemente e se il sistema integrato soddisfa le specifiche di progetto. L'obiettivo è garantire che, quando le unità sono combinate, il software funzioni senza intoppi e risponda alle esigenze dell'applicazione in modo coerente.

3.3.4.3 Test di *sistema$_G$*

I test di *sistema$_G$* vengono definiti durante la fase di analisi dei requisiti con l'obiettivo di misurare la copertura dei requisiti derivati dal capitolato d'appalto. Questa fase del processo di software testing mira a verificare l'intero *sistema$_G$* come entità unificata. Durante il test di *sistema$_G$*, il software viene valutato nella sua completezza, con l'obiettivo di accertare che tutte

le componenti integrate funzionino insieme in modo coerente per soddisfare gli obiettivi e i requisiti del *sistema*$_G$. Per la definizione dei test si rimanda al *Piano di Qualifica v1.0*.

3.3.4.4 Test di accettazione

Il superamento dei test di accettazione è fondamentale poiché dimostra il soddisfacimento della Proponente rispetto al prodotto software. L'esito positivo di questi test fornisce l'approvazione finale e la conferma che il software è conforme alle aspettative. Solo dopo il superamento dei test di accettazione il team *SWAT Engineering* potrà procedere con il rilascio ufficiale del prodotto, garantendo che sia pronto all'utilizzo nell'ambiente operativo previsto. Per la definizione dei test si rimanda al *Piano di Qualifica v1.0*.

3.3.5 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.

3.3.6 Stato dei test

Nella sezione relativa ai test nel *Piano di Qualifica v1.0* 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 Validazione

3.4.1 Descrizione e scopo

Il processo di validazione è un processo di supporto il cui obiettivo primario è ottenere la conferma che il software sviluppato rispetti le aspettative della Proponente. Lo scopo principale della validazione è quindi garantire che il software sia implementato in maniera adeguata e che funzioni correttamente nel contesto operativo previsto, fornendo un prodotto finale soddisfacente e conforme alle esigenze della Proponente. Inoltre, durante la validazione, devono essere soddisfatti tutti i requisiti previamente definiti, assicurando che il software risponda pienamente alle specifiche e alle richieste stabilite.

3.4.2 Test di accettazione

I test di accettazione sono condotti dai Verificatori in presenza della Proponente per assicurarsi che il software soddisfi i requisiti e le aspettative della Proponente stessa. Per la definizione dei vari test si rimanda al *Piano di Qualifica v2.0*. Gli eventuali problemi riscontrati durante i test vengono documentati e risolti prima che il software venga rilasciato ufficialmente. L'obiettivo finale dei test di accettazione è ottenere l'approvazione della Proponente, confermando che il software è pronto per essere utilizzato in produzione. Prima di effettuare i test di accettazione, i Verificatori hanno la responsabilità di eseguire i test di sistema in un

ambiente identico a quello di installazione: è precondizione necessaria del collaudo che i test di sistema diano esito positivo.

3.5 Gestione della qualità

3.5.1 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.

3.5.2 Obiettivi

La gestione della qualità si propone di raggiungere i seguenti 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 v1.0*, in cui sono presenti varie metriche con le relative soglie di valori accettabili ed ideali.

3.5.3 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.

3.5.4 Struttura metriche

Nel *Piano di Qualifica v1.0* le varie metriche sono descritte mediante delle tabelle organizzate secondo la seguente struttura:

- **Metrica**: il codice della metrica seguendo il codice identificativo;
- **Descrizione**: il nome della metrica;
- **Valore di accettazione**: valore considerato accettabile;
- **Valore ideale**: valore ideale che dovrebbe essere assunto dalla metrica.

3.5.5 Grafici metriche

Per realizzare i grafici relativi alle metriche e all'andamento del progetto, si usa Google Sheets. La descrizione delle metriche usate si trova nella Sezione 5.

3.5.6 Metriche

La definizione delle seguenti metriche si può trovare nella Sezione 5.1.4.

Metrica	Descrizione
MNS	Metriche Non Soddisfatte

Table 6: Metriche relative alla gestione della qualità

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*$_G$ fino alla conclusione del progetto. Questo implica una distribuzione coerente del lavoro in linea con le scadenze fissate per le revisioni *RTB*$_G$, *PB*$_G$ e *CA*$_G$. 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*$_G$ in periodi di tempo specifici.

Oltre a definire gli obiettivi per ogni *sprint*$_G$, 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 deve sempre rimanere aggiornata e sensata rispetto all'andamento generale del progetto e agli obiettivi imminenti. Queste previsioni vengono formalizzate nel *Piano di Progetto v1.0*, che diventa un punto di riferimento durante l'evento di *sprint planning*$_G$ per definire gli obiettivi del successivo *sprint*$_G$.

Eventuali variazioni nella distribuzione dei ruoli, rispetto alla pianificazione iniziale, vengono documentate e giustificate nel **Consuntivo** del *Piano di Progetto v1.0*. Questo approccio consente al team di adattarsi in modo flessibile alle esigenze emergenti, mantenendo costantemente un quadro chiaro delle variazioni e delle ragioni che sottendono a tali modifiche.

4.1.2.2 *Documentazione*$_G$ fine *sprint*$_G$

Il Responsabile è incaricato di sviluppare un preventivo dettagliato per le successive due settimane subito dopo la conclusione dello *sprint*$_G$ attuale, fornendo una guida chiara al team su aspettative e obiettivi.

Nel contesto della gestione dello *sprint*$_G$, il Responsabile si impegna anche a compilare un consuntivo dettagliato dello sprint appena concluso.

Per garantire un monitoraggio efficace delle prestazioni, il Responsabile utilizza e calcola le metriche, presenti nel documento *Piano di Qualifica v1.0*, rispetto allo *sprint*$_G$ appena concluso, valutando l'efficacia del processo e apportando eventuali correzioni o miglioramenti per il successo continuo del progetto.

4.1.2.3 Preventivo

Per utilizzare efficacemente lo script per il calcolo preventivo, si procede nel seguente modo:

- Eseguire lo script init_preventivo.py, seguendo attentamente le istruzioni fornite. Questo script guida la creazione dei file CSV necessari, che sono utilizzati per inserire i dati relativi a due preventivi successivi;

- Accedere al file preventivonumerosprint.csv situato nella directory preventivi/preventivi_csv/. In questo file, si inserisce, con precisione, i dati riguardanti le ore dedicate da ciascun membro del team, specificando le risorse allocate per i due preventivi successivi;
- Eseguire lo script auto_preventivo.py, indicando lo *sprint*$_G$ per il quale si desidera generare il preventivo. L'esecuzione di questo script applica automaticamente i dati inseriti, producendo un preventivo accurato per lo *sprint*$_G$ selezionato;
- All'interno del file typst, dove si intende inserire il preventivo, utilizzare le funzioni prospettoOrario e prospettoEconomico fornite nel modulo functions.typ. Queste funzioni consentono di integrare facilmente i dati di prospetto orario ed economico nel documento, facilitando la visualizzazione e la gestione delle informazioni relative al preventivo.

4.1.2.4 Consuntivo

Si analizzano le attività svolte, i tempi impiegati rispetto alle stime previste e qualsiasi deviazione dai piani originali. Tale consuntivo rappresenta una preziosa fonte di apprendimento per il team, consentendo di identificare aree di miglioramento e di ottimizzare la pianificazione futura.

È stato sviluppato uno script in *Python*$_G$ (timeresource_sheets_downloader.py), integrato con *Typst*$_G$, per semplificare la generazione automatica di tabelle e grafici relativi al consuntivo di ogni singolo *sprint*$_G$ del progetto. L'automazione coinvolge l'estrazione dei dati dal foglio di calcolo condiviso, dove vengono registrate le ore produttive del team. I dati vengono successivamente organizzati per *sprint*$_G$ e archiviati in file CSV.

L'elaborazione comprende la creazione di un DataFrame consolidato che rappresenta le ore lavorate per ogni ruolo e membro del team durante uno *sprint*$_G$, includendo anche i costi associati. I risultati di questa analisi vengono salvati in un file CSV.

Per visualizzare i dati aggiornati relativi a ciascuno *sprint*$_G$, è possibile utilizzare le seguenti funzioni all'interno del file:

```
\#rendicontazioneOreAPosteriori(sprintNumber: "number")
\#rendicontazioneCostiAPosteriori(sprintNumber: "number")
```

Dove "number" rappresenta il numero a 2 cifre, specifico dello *sprint*$_G$ di interesse. Queste funzioni consentono di ottenere report aggiornati sulla distribuzione delle ore per ruolo e persona, nonché sui costi associati, semplificando la gestione.

4.1.3 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:

4.1.3.1 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*$_G$ 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.

Dopo l'*RTB*$_G$ il diario di bordo si tiene ogni venerdì alle 15, pertanto il Responsabile è tenuto a creare le slide anche per il successivo diario di bordo, mentre chi lo presenta è il nuovo Responsabile, eletto nell'incontro interno del Venerdì.

4.1.3.2 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;
- Stesura e avanzamento del documento *Piano di Qualifica*.

4.1.3.3 Analista

Figura professionale che si occupa di analizzare, comprendere e definire i requisiti e le specifiche di un *sistema*$_G$ 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*$_G$;
- Raccolta dei requisiti per definire le necessità e le funzionalità richieste;
- Stesura del documento *Analisi dei Requisiti*;
- Creazione diagrammi *UML*$_G$;

4.1.3.4 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*$_G$, 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*$_G$;
- 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*.

4.1.3.5 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 manutenibile;
- Creazione di test per la verifica del software;
- Ampliamento delle *Specifiche Tecniche* conforme alle esigenze del progetto;
- Risoluzione di *bug*$_G$ e problemi di performance;
- Realizzazione del *Manuale Utente*;
- Collaborazione con il team per l'integrazione del codice e il mantenimento della coerenza del progetto.

4.1.3.6 Verificatore

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

I suoi compiti:

- Revisione e valutazione della *documentazione*$_G$ 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.

4.1.4 Cambio dei ruoli

Per garantire che ogni membro svolga tutti i ruoli menzionati per un periodo di tempo consono, il team si impegna a cambiarli ogni settimana.

4.1.5 Tracciamento del tempo speso

Al fine di tracciare il tempo speso nel corso del progetto, nei diversi ruoli, si utilizza uno *spreadsheet*$_G$ 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.

4.1.6 Gestione strumenti coordinamento

4.1.6.1 Ticketing

L'Amministratore crea, durante gli incontri interni, e gestisce le varie task in modo tale ogni membro del team sia informato su ciò che ci si propone di fare durante lo *sprint*$_G$ e sia in grado di verificare il progresso effettuato in qualsiasi momento.

Si segue il seguente metodo:
1. L'Amministratore crea la issue su Github;
2. Assegna il task al membro, con la relativa priorità e grandezza;

3. Imposta una data di inizio e fine del task e lo sposta nella *corsia*$_G$ Ready;
4. Assegna l'issue alla *milestone*$_G$ dello *sprint*$_G$ attuale;
5. Il membro incaricato della issue la sposta nella *corsia*$_G$ "In Progress" nel momento in cui inizia a lavorarci;
6. Una volta concluso il task, la issue viene spostata nella *corsia*$_G$ "Ready To Review";
7. Successivamente il Verificatore segue i passi definiti nella Sezione 3.3.2.1;
8. La issue viene chiusa in automatico, se linkata ad una pull request che viene mergiata nel ramo principale main, altrimenti si procede manualmente.

Ogni issue è composta da:

- **Titolo**: titolo significativo;
- **Descrizione**: una breve descrizione o checklist delle cose da fare;
- **Assignee**: membro/membri incaricati;
- **Priority**: la priorità, può essere:
  1. Low;
  2. Medium;
  3. High;
  4. Urgent.
- **Size**: la grandezza del lavoro da svolgere:
  1. Tiny;
  2. Small;
  3. Medium;
  4. High;
  5. Large.
- **Intervallo temporale**: una data di inizio e una di fine;
- ***Milestone*$_G$**: *milestone*$_G$ associata alla issue, se il completamento non è possibile entro lo *sprint*$_G$ attuale, è necessario spostare l'issue nella *milestone*$_G$ successiva.

4.1.7 Gestione dei rischi

La gestione dei rischi è un approccio sistematico finalizzato a individuare, valutare e affrontare le possibili minacce o opportunità che possono influenzare il successo di un progetto. Questo processo coinvolge l'identificazione dei rischi, la valutazione della loro probabilità e impatto, la pianificazione di strategie di mitigazione o sfruttamento e il monitoraggio continuo durante l'intero ciclo di vita del progetto.

4.1.7.1 Struttura dei rischi

I rischi vengono suddivisi in tre categorie:

- Rischi tecnologici;
- Rischi comunicativi;
- Rischi pianificazione.

Ogni rischio viene identificato da un codice univoco avente la seguente struttura:

R[Tipologia][Numero] - [Nome]

Legenda:
- **Tipologia**: il tipo del rischio, può essere:
  - T se tecnologico;
  - C se comunicativo;

  - P se di pianificazione.
- **Numero**: è un numero progressivo univoco per ogni rischio della sua tipologia;
- **Nome**: nome del rischio.

4.1.8 Comunicazione e incontri

4.1.8.1 Gestione delle comunicazioni

4.1.8.1.1 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 funzionalità offerte da servizi esterni quali GitHub.

4.1.8.1.2 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.

4.1.8.2 Gestione degli incontri

4.1.8.2.1 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 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*$_G$ quotidiani (*Daily Scrum*$_G$) di 15 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.

Dopo la sessione invernale degli esami, gli *stand-up meeting*$_G$ quotidiani si svolgono virtualmente.

4.1.8.2.2 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*$_G$. 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 tec-

nologia, imparare le *best practices*$_G$ e acquisire competenze più avanzate. Il Responsabile ha il compito della stesura del verbale esterno, che viene successivamente convalidato, con firma, dalla Proponente.

4.1.8.2.3 Doveri dei partecipanti

I partecipanti hanno il dovere di:

- Arrivare puntuali alle riunioni;
- Comunicare al Responsabile, il prima possibile, eventuali ritardi o assenze;
- Partecipare attivamente agli argomenti dell'*ordine del giorno*$_G$
- Mantenere un comportamento corretto durante tutta la riunione.

4.2 Formazione

4.2.1 Scopo e aspettative

Il processo è progettato per normare il modo in cui il team *SWAT Engineering* acquisisce conoscenze relative alle tecnologie richieste per la produzione dei documenti e del prodotto. L'attività di formazione è progettata per garantire che ogni membro del team raggiunga una competenza sufficiente nell'utilizzo consapevole delle tecnologie selezionate per la realizzazione del progetto.

4.2.2 Formazione individuale dei membri

Ciascun membro del gruppo assume la responsabilità di approfondire individualmente le diverse tecnologie necessarie per lo sviluppo del prodotto. Per facilitare il processo di apprendimento collettivo, si promuove attivamente la condivisione di conoscenze tra i membri, permettendo a coloro che possiedono una maggiore competenza di condividere le proprie nozioni con i colleghi meno esperti.

4.2.3 Guide e documentazione

Si riporta la lista delle guide e documentazione da prendere come riferimento:

- ***Typst***$_G$: https://typst.app/docs/ (22-02-2024);
- **Github**: https://docs.github.com/en
- ***Python***$_G$: https://docs.python.org/3/ (22-02-2024);
- ***Grafana***$_G$: https://grafana.com/docs/grafana/latest/ (22-02-2024);
- **Apache *Kafka***$_G$: https://kafka.apache.org/20/documentation.html (22-02-2024);
- ***Clickhouse***$_G$: https://clickhouse.com/docs (22-02-2024).

Questa lista non intende essere forzata, ciascun membro è libero di approfondire le proprie conoscenze utilizzando materiale aggiuntivo. Qualora lo ritenesse utile ed esaustivo è pregato di condividerlo con gli altri membri.

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: $\mathrm{EV} = \text{Percentuale di completamento del lavoro} \cdot \mathrm{BAC}$

  in cui la percentuale di completamento del lavoro è misurata con le ore produttive effettive degli *sprint*$_G$; in particolare la percentuale è costituita dal rapporto tra le ore effettive e le ore produttive totali a disposizione (570)
- **PV**: Planned Value - il valore del lavoro pianificato fino a un dato momento

  Formula: $\mathrm{PV} = \text{Percentuale di pianificazione del lavoro} \cdot \mathrm{BAC}$

  in cui la percentuale di pianificazione del lavoro è misurata con le ore preventivate per gli *sprint*$_G$; in particolare la percentuale è costituita dal rapporto tra le ore preventivate e le ore produttive totali a disposizione (570)
- **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: $\mathrm{CPI} = \frac{\mathrm{EV}}{\mathrm{AC}}$;
- **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: $\mathrm{SV} = \mathrm{EV} - \mathrm{PV}$;
- **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

- **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.

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.

  Per calcolarlo si usa uno strumento online: https://farfalla-project.org/readability_static/ (22-02-2024);

- **CO**: Correttezza Ortografica - numero errori grammaticali ed ortografici in un documento.

5.1.4 Gestione della qualità

- **MNS**: Metriche Non Soddisfatte - Numero delle metriche che il progetto non riesce a raggiungere o mantenere.

5.2 Metriche per la qualità di prodotto

5.2.1 Funzionalità

- **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 percentuale 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: $\mathrm{ROPS} = \frac{\text{requisiti opzionali soddisfatti}}{\text{requisiti opzionali totali}} \cdot 100$;

5.2.2 Usabilità

- **FU**: Facilità di Utilizzo - quantità di click che l'utente deve effettuare per raggiungere un obiettivo desiderato;

- **TA**: Tempo di Apprendimento - il tempo richiesto da una persona per comprendere e padroneggiare l'uso del *sistema*$_G$.

5.2.3 Affidabilità

- **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: $\mathrm{PTCP} = \frac{\text{test superati}}{\text{test totali}} \cdot 100$;

- **CC**: Code Coverage - numero di linee di codice convalidate con successo nell'ambito di una procedura di test

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

- **BC**: Branch Coverage - valuta quanti rami condizionali nel codice sorgente sono stati eseguiti durante l'esecuzione dei test. Formula: $\mathrm{BC} = \frac{\text{Numero di rami condizionali eseguiti}}{\text{numero totale dei rami condizionali}} \cdot 100$;

- **SC**: Statement Coverage - valuta quanti statement o istruzioni del codice sorgente sono stati eseguiti durante l'esecuzione dei test. Formula: $\mathrm{SC} = \frac{\text{numero statement eseguiti}}{\text{numero totale statement}} \cdot 100$.

5.2.4 Manutenibilità

- **SFIN**: Structure Fan In - rappresenta il numero di moduli o componenti direttamente collegati o dipendenti da un modulo o una funzione specifica; un fan-in elevato indica che molte parti del *sistema*$_G$ dipendono da un particolare modulo;
- **SFOUT**: Structure Fan Out - rappresenta il numero di dipendenze o connessioni che un componente o modulo particolare ha con altri componenti o moduli. Misura quanti altri elementi dipendono o interagiscono con un dato elemento all'interno di un *sistema*$_G$; un fan-out elevato può indicare che un modulo ha molte dipendenze da altri moduli;
- **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: $\mathrm{CCM} = e - n + 2$, 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.

5.2.5 Efficienza

- **CPUU**: Maximum CPU usage - indica il picco massimo di utilizzo della CPU durante un determinato periodo di tempo. Utilizzata per valutare il carico massimo di lavoro che il *sistema*$_G$ può gestire e per identificare possibili problemi di performance;
- **RAMU**: Maximum RAM usage - indica il picco massimo di utilizzo della memoria RAM durante un determinato periodo di tempo.Importante per valutare l'efficienza nell'uso della memoria del sistema e per identificare eventuali situazioni di sovrautilizzo della memoria che potrebbero causare rallentamenti o crash;
- **TDE**: Tempo Di Elaborazione - rappresenta il tempo trascorso tra il momento in cui i dati sono generati e il momento in cui vengono visualizzati nella *dashboard*$_G$ di *Grafana*$_G$.