Every line of code is worth the SWEat

Norme di Progetto

27 marzo 2025

Uso	Interno
Destinatari	Gruppo SWE@
Responsabile	Davide Picello
Redattori	Andrea Precoma
	Davide Marin
	Davide Martinelli
	Davide Picello
	Klaudio Merja
	Riccardo Milan
Verificatori	Andrea Perozzo
	Andrea Precoma
	Davide Marin
	Davide Martinelli
	Davide Picello
	Klaudio Merja
	Riccardo Milan

Registro delle modifiche

Ver.	Data	Redattori	Verificatori	Descrizione
2.0.0	27/03/2025	Andrea Precoma	Klaudio Merja	• Aggiornate strutture delle *repository*
1.3.0	27/03/2025	Andrea Precoma	Klaudio Merja	• Rinominato MPD-ROPS in MPD-RFS • Rimosse metriche MPD-SC, MPD-FD, MPD-COC, MPD-SFI, MPD-SFO e MPD-RM
1.2.0	03/03/2025	Andrea Precoma	Davide Marin	• Aggiornata sezione Piano di Progetto • Aggiornata norma sui riferimenti esterni accompagnati dall'ultimo accesso • Corretta fase di approvazione • Aggiunti Manuale Utente e Specifica Tecnica nella documentazione fornita
1.1.0	11/02/2025	Andrea Precoma	Andrea Perozzo	• Rimozione identificativo ORG per attività organizzative • Rimozione del registro delle modifiche e riassunto per i verbali • Cambiamento della nomenclatura dei verbali
1.0.1	05/02/2025	Klaudio Merja	Andrea Precoma	• Correzione dei riferimenti alla documentazione del gruppo
1.0.0	26/01/2025	Davide Marin	Andrea Perozzo	Approvazione versione finale del documento per rilascio in RTB
0.9.0	24/01/2025	Davide Marin	Andrea Perozzo Davide Martinelli	• Aggiunta descrizione della *repository* NearYou • Ultimazione di alcuni paragrafi
0.8.0	12/01/2025	Davide Marin	Davide Martinelli Davide Picello	• Aggiunte metriche per la qualità di prodotto
0.7.0	8/01/2025	Davide Martinelli	Davide Marin Davide Picello	• Aggiunte metriche per la qualità di processo • Stesura sezione «Standard per la qualità» • Ampliamento sezioni «Verifica» e «Validazione»
0.6.0	22/12/2024	Davide Picello	Davide Marin, Klaudio Merja	• Scrittura della sezione «Casi d'uso» in «Processi primari»
0.5.0	22/12/2024	Riccardo Milan	Davide Marin Klaudio Merja	• Aggiunta del capitolo «Processi organizzativi» • Correzioni di stile
0.4.0	30/11/2024	Andrea Precoma	Riccardo Milan	• Modifica del *versioning* • Modifica delle regole di stile • Aggiunta struttura della *repository* NearYou • Correzione ortografica e di stile
0.3.0	27/11/2024	Davide Picello	Davide Marin	• Scrittura del capitolo «Processi primari»
0.2.0	26/11/2024	Andrea Precoma	Davide Marin Davide Picello	• Struttura della sezione «Processi di supporto» • Stesura dei principali paragrafi

Ver.	Data	Redattori	Verificatori	Descrizione
0.1.0	14/11/2024	Klaudio Merja	Andrea Precoma	• Struttura e introduzione del documento

Indice

Elenco delle tabelle

1. Introduzione

1.1. Scopo del documento

Lo scopo principale del documento Norme di Progetto è quello di documentare il *way of working*[g] che deve essere adottato dai membri del gruppo per garantire la coerenza nel lavoro svolto.

Il documento adotta un approccio di tipo incrementale, ovvero è frutto di una serie di fasi di modifica che avvengono durante il suo ciclo di vita. È quindi soggetto durante lo svolgimento del progetto didattico a continui aggiornamenti, a seguito delle decisioni prese dal gruppo, volti a migliorare il *way of working* stesso. I membri del gruppo sono quindi tenuti a prendere visione del documento ogni qual volta esso sia oggetto di modifiche.

1.2. Scopo del progetto didattico

Lo scopo principale del progetto *NearYou - Smart custom advertising platform* proposto dall'azienda Sync Lab S.r.l. è quello di sviluppare un prodotto che sfrutti la GenAI[g] per la creazione di pubblicità mirate sui singoli utenti, utilizzando all'interno di *prompt*[g] dati come la posizione in tempo reale, le informazioni personali e i dati di profilazione degli utenti stessi.

L'obiettivo del prodotto è quello di rendere le campagne pubblicitarie delle aziende interessate il più personalizzate e ottimizzate possibili ed aumentare il coinvolgimento dell'utente finale, con lo scopo di ridurre la disconnessione tra messaggio e destinatario e portare un miglioramento sul ROI[g] della campagna stessa.

1.3. Glossario

Per evitare eventuali ambiguità e incomprensioni sulla terminologia adottata nella documentazione redatta dal gruppo, viene fornito un glossario.

La prima occorrenza di un termine definito all'interno del glossario presente all'interno di un documento viene sottolineato e seguito dalla lettera «g» posta ad apice (e.g. termine[g]).

1.4. Riferimenti

1.4.1. Riferimenti normativi

- Regolamento del progetto didattico, *slide* 7, 12, 13, 19 e 23 (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf
- ISO/IEC 12207:1995, da pagina 9 a 47 (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

1.4.2. Riferimenti informativi

- Glossario (v2.0.0)
  https://sweatunipd.github.io/docs/pb/glossario_ver2.0.0.pdf
- Capitolato C4 - Sync Lab S.r.l. (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C4.pdf
- Lezione T07 - Qualità del software, da *slide* 7 a 17 (ultimo accesso in data 27/03/205)
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T07.pdf
- Lezione T08 - Qualità di processo, da *slide* 2 a 7 (ultimo accesso in data 27/03/2025)
  https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T08.pdf

2. Processi primari

Lo *standard* ISO/IEC 12207:1995, riferimento internazionale per i processi del ciclo di vita del *software*, definisce come processi primari le attività di: acquisizione, fornitura, sviluppo, gestione operativa e manutenzione.

Tuttavia, data la natura didattica del progetto, saremo tenuti ad occuparci solo delle parti di fornitura e sviluppo.

2.1. Fornitura

Il processo di fornitura contiene le attività e i compiti del fornitore, il quale dovrà accordarsi con il proponente per stabilire ufficialmente, tramite un contratto, i vari vincoli e requisiti del progetto. Successivamente si procede con la redazione del documento Piano di Progetto, il quale avrà il compito di pianificare e monitorare le attività da svolgere durante i periodi del progetto.

Le attività che compongono questo processo sono:
- Iniziazione
- Preparazione della risposta
- Contratto
- Pianificazione
- Esecuzione e controllo
- Revisione e valutazione
- Consegna e completamento

2.1.1. Comunicazioni con il proponente

L'azienda si è resa raggiungibile, per via testuale, tramite *e-mail* per le comunicazioni ufficiali e tramite la piattaforma di messaggistica istantanea Discord, invece, per le comunicazioni più veloci. È stato deciso inoltre di utilizzare l'applicazione di teleconferenza Google Meet per le riunioni in cui è prevista la presenza di tutto il gruppo. L'azienda si è resa disponibile anche ad accogliere il gruppo nei loro uffici.

Per ogni incontro, verrà redatto un verbale riguardante le discussioni e decisioni avvenute nello stesso. Questi verbali, dopo essere stati approvati e firmati dall'azienda, saranno disponibili presso il sito *web* del gruppo **https://sweatunipd.github.io** (ultimo accesso 03/03/2025).

2.1.1.1. SAL

È stato inoltre concordato un incontro settimanale di circa 20-30 minuti, tendenzialmente il mercoledì pomeriggio, riguardo allo Stato di Avanzamento dei Lavori (SAL[g]). In questi incontri il gruppo è tenuto a esporre i progressi e i dubbi dell'ultima settimana di lavoro, confrontandosi con il proponente e delineando le attività da svolgere nella settimana a venire.

2.1.1.2. Sessioni di deep dive tecnologico

Il proponente si è anche reso disponibile per organizzare delle sessioni di deep dive[g] tecnologico per approfondire l'utilizzo di alcune tecnologie che potrebbero rivelarsi particolarmente ostiche per il gruppo. Queste verranno fissate quando necessarie.

2.1.2. Documentazione fornita

Di seguito viene riportato l'elenco dei documenti che il gruppo consegnerà all'azienda proponente Sync Lab S.r.l. e ai committenti, nonché i prof. Tullio Vardanega e prof. Riccardo Cardin.

2.1.2.1. Analisi dei Requisiti

L'Analisi dei Requisiti è un documento redatto dagli analisti che definisce le funzionalità necessarie per soddisfare le richieste del proponente.

2.1.2.1.1. Casi d'uso

In questo documento rientra la realizzazione dei casi d'uso (*use case*): descrizioni formali di come un sistema *software* interagisce con utenti o con altre entità esterne al sistema, chiamati attori, per raggiungere un obiettivo specifico. Questi sono composti da una descrizione testuale e dal loro diagramma, essi sono usati per documentare le varie funzionalità attese dal sistema ad alto livello, quindi non specificando i dettagli implementativi.

Ogni caso d'uso deve comprendere le seguenti informazioni:

2.1.2.1.1.1. Identificazione

Ogni caso d'uso deve essere identificato da un codice univoco, secondo la nomenclatura **UC[numero_use_case]**, un trattino ed un breve titolo esplicativo.

2.1.2.1.1.2. Attori

Gli attori rappresentano entità esterne che interagiscono con il sistema. Ogni caso d'uso specifica una funzionalità creata per un determinato attore.

Esistono più tipi di attori:

- **Attore primario**: L'utente o *stakeholder*[g] che avvia il caso d'uso ed ha un obiettivo da raggiungere utilizzando il sistema.
- **Attore di supporto**: Un utente o *stakeholder* coinvolto nel caso d'uso, ma che non inizia l'interazione con il sistema. Può fornire ulteriori *input* o risorse necessarie per completare lo *use case*. La sua presenza non è obbligatoria all'interno di un caso d'uso.

2.1.2.1.1.3. Precondizioni

Condizioni che devono essere vere o soddisfatte affinché il caso d'uso possa iniziare. Definiscono lo stato iniziale del sistema ed i requisiti che devono essere garantiti affinché il caso d'uso sia valido.

2.1.2.1.1.4. Postcondizioni

Stato finale del sistema dopo che il caso d'uso è stato completato. Utili per verificare che l'obiettivo sia stato raggiunto e per documentare le modifiche apportate al sistema.

2.1.2.1.1.5. Scenario principale

Lo scenario principale descrive la sequenza di azioni *standard* che l'attore ed il sistema eseguono per raggiungere l'obiettivo del caso d'uso. Include tutti i passi principali in ordine sequenziale.

2.1.2.1.1.6. Relazioni

Nei casi d'uso possiamo avere i seguenti tipi di relazione tra attore e caso d'uso.

2.1.2.1.1.6.1. Associazione

Congiunge semplicemente gli attori con i casi d'uso a cui prendono parte. Un attore può essere associato a qualsiasi numero di casi d'uso e viceversa. L'associazione implica uno scambio di informazioni tra attore e caso d'uso associato. Viene indicata nel diagramma con una linea nera.

2.1.2.1.1.6.2. Generalizzazione

Riguarda sia gli attori che i casi d'uso e rappresenta una relazione tra una classe, un caso d'uso o un attore più generale (superclasse) ed uno più specifico (sottoclasse). Questo tipo di relazione segue il principio dell'ereditarietà, in cui la sottoclasse eredita le caratteristiche dalla superclasse e può aggiungere o modificare specifici dettagli. La generalizzazione si rappresenta con una linea che collega le due entità (classi, attori o casi d'uso) e termina con un triangolo vuoto sulla punta, che indica la direzione verso la classe o entità più generale.

2.1.2.1.1.6.3. Inclusione

Le inclusioni rappresentano funzionalità o comportamenti comuni che possono essere riutilizzati in più casi d'uso per evitare duplicazioni, inoltre indicano che l'esecuzione di una funzione implica l'esecuzione di tutte le sue inclusioni. Viene indicata nel diagramma come una freccia tratteggiata con indicato "«*include*»". La funzione alla base della freccia include completamente la funzione alla punta della freccia.

2.1.2.1.1.6.4. Estensione

Le estensioni definiscono variazioni opzionali o eccezioni al comportamento principale del caso d'uso. Questi flussi si attivano solo se si verificano determinate condizioni. Viene indicata nel diagramma come una freccia tratteggiata con indicato "«*extend*»". La funzione alla base della freccia può essere impiegata nel contesto della funzione alla punta della freccia. Indichiamo esplicitamente anche la condizione per l'estensione e l'*extension point*, che identifica il punto nel caso d'uso di base in cui è possibile inserire il comportamento del caso d'uso esteso.

2.1.2.1.1.7. Diagrammi dei casi d'uso

I diagrammi dei casi d'uso rappresentano graficamente i casi d'uso, gli attori e le relazioni tra loro, rendendo più intuitivo comprendere le interazioni tra il sistema e i suoi attori.

Gli elementi grafici principali da utilizzare in questi diagrammi sono:
- **Sistema**: Il sistema è rappresentato da un rettangolo vuoto al cui interno vengono inseriti gli elementi grafici rappresentanti le sue caratteristiche. Gli elementi che invece rappresentano entità esterne sono posizionati all'esterno del rettangolo.
- **Attore**: Rappresentato graficamente da un'icona raffigurante un omino stilizzato con la relativa etichetta univoca.
- **Caso d'uso**: Rappresentato graficamente come un'ellisse contenente il nome del caso d'uso. Concretamente rappresenta una funzione o servizio offerto dal sistema ad uno o più attori.
- **Associazione**: Relazione tra attori e casi d'uso. Essa implica uno scambio di informazioni tra attore e caso d'uso associato.

2.1.2.2. Piano di Progetto

Il Piano di Progetto è un documento il cui scopo è quello di definire in modo dettagliato l'organizzazione, le attività, le risorse, i tempi e i criteri necessari per completare con successo il progetto.

Viene redatto e aggiornato dal responsabile, con il supporto degli amministratori, durante l'intera durata del progetto.

Il documento conterrà le seguenti informazioni:
- **Analisi dei rischi**: Utile a individuare le possibili difficoltà che il gruppo può incontrare.
- **Modello di sviluppo**: Descrive quale modello di sviluppo è stato scelto ed adottato dal gruppo per lo svolgimento del progetto.
- **Pianificazione**: Contiene ed espone in quale modo il gruppo ha deciso di pianificare le attività da svolgere e la loro suddivisione in periodi. Per ogni periodo inoltre saranno indicati:
  - **Introduzione del periodo**: Breve introduzione con informazioni chiave come data di inizio e fine periodo e i rischi attesi all'interno dello stesso.
  - **Attività**: Descrive le attività pianificate che dovranno essere svolte entro il termine del periodo. Devono essere accompagnate da una descrizione esaustiva in funzione di guida per chi dovrà svolgerle.
  - **Preventivo**: Contiene il ruolo e il preventivo orario per ciascun membro del gruppo.

- **Consuntivo**: Viene redatto alla fine di ogni periodo, contiene il numero di ore effettivamente impiegato da ogni membro del gruppo, utile per avere un riscontro del lavoro svolto ed effettuare analisi sull'efficienza.
- **Risorse rimanenti e preventivo a finire**: Indica quante ore (e i relativi costi) sono rimaste a disposizione per la conclusione del progetto. Sono accompagnate da una colonna che indica la percentuale di risorse consumate rispetto al totale pianificato per ogni ruolo, così da inquadrare meglio la distribuzione delle ore.
- **Retrospettiva**: Sezione redatta a fine periodo, contiene una descrizione testuale delle difficoltà riscontrate e consigli utili da adottare per il lavoro futuro. Si struttura nei seguenti paragrafi:
  - **Cosa ha funzionato**: Aspetti positivi del periodo concluso.
  - **Cosa non ha funzionato**: Difficoltà che hanno rallentato il lavoro.
  - **Cosa fare per migliorare**: Considerazioni migliorative per gli *sprint*[g] successivi.
  - **Rischi riscontrati**: Ricapitolazione dei rischi pianificati, analizzando se ve ne sono verificati di nuovi, e come sono stati affrontati.
  - **Attività future**: Lavori da cominciare nel periodo successivo che derivano da quello appena concluso, utile per guidare la pianificazione successiva.

2.1.2.3. Piano di Qualifica

Il Piano di Qualifica è un documento formale che definisce le strategie, le attività e gli approcci necessari per garantire la qualità del prodotto *software* durante tutto il corso del progetto. Descrive le modalità di verifica e validazione, gli *standard* adottati e le procedure di qualità da seguire.

Il suo scopo principale è assicurare che il prodotto finale sia conforme alle specifiche richieste e alle aspettative del committente, monitorando il progresso rispetto agli obiettivi prefissati. Ogni membro del *team* coinvolto nello sviluppo farà riferimento a questo documento per mantenere e garantire i livelli di qualità stabiliti.

Al suo interno, sono presenti le seguenti informazioni:
- **Introduzione**: Descrizione dello scopo del documento e del prodotto da noi sviluppati, contiene inoltre i riferimenti normativi e informativi utili per la stesura e lettura del documento.
- **Obiettivi di qualità**: Sezione che definisce gli obiettivi di qualità posti dal gruppo, si suddivide in:
  - **Qualità di processo**: Descrizione di metriche e parametri da seguire per rispettare gli obiettivi di qualità per i processi.
  - **Qualità di prodotto**: Descrizione di metriche e parametri da seguire per rispettare gli obiettivi di qualità per il prodotto finale.
- **Specifiche dei test**: Descrive le attività di *testing* che saranno effettuate per garantire il rispetto dei vincoli definiti nei requisiti del progetto.
- **Cruscotto di monitoraggio della qualità**: Contiene una descrizione più dettagliata di ogni metrica, il suo scopo e come ricavarla.

2.1.2.4. Manuale Utente

Il Manuale Utente descrive nel dettaglio come utilizzare il prodotto *software*. Vengono elencati i prerequisiti, le operazioni da svolgere per installare, avviare e terminare l'applicativo, le istruzioni per accedere e usufruire dei servizi forniti. Ogni passaggio è accompagnato da immagini, comandi o *link* a documentazione esterna.

2.1.2.5. Specifica Tecnica

Il documento descrive accuratamente gli aspetti progettuali e le scelte tecnologiche. Deve quindi trattare le seguenti scelte:
- **Tecnologie adottate** motivate da ragionamenti e descrizioni nei casi d'uso del prodotto.
- **Architettura logica** per componenti, ruoli, connessioni e interazioni.

- **Architettura di *deployment*** per l'allocazione delle parti nel sistema in esecuzione.
- ***Design pattern* architetturali** adottati per risolvere alcuni problemi e agevolare la struttura del codice.

Il documento deve essere accompagnato anche dai diagrammi delle classi e, se necessario, da porzioni di codice.

2.1.2.6. Glossario

Il Glossario è un documento di supporto che raccoglie i termini tecnici e specifici utilizzati all'interno del progetto, fornendone definizioni chiare e univoche. È destinato a tutti gli *stakeholder*, i membri del *team*, i committenti e l'azienda proponente, con l'obiettivo di prevenire ambiguità e incomprensioni. Questo strumento è fondamentale per migliorare la coerenza e la qualità della documentazione prodotta.

2.1.2.7. Lettera di presentazione

La Lettera di Presentazione è un documento formale attraverso il quale il gruppo esprime la propria volontà di partecipare alla fase di revisione del progetto di una determinata *milestone*. Include la documentazione prodotta dal gruppo e, se necessario, un aggiornamento sui costi e sulle tempistiche.

La lettera sottolinea l'impegno del *team* nel rispettare le scadenze e le aspettative fissate.

2.1.3. Strumenti

Vengono riportati di seguito tutti gli strumenti adottati per il processo di fornitura.

2.1.3.1. Discord

Piattaforma di messaggistica in tempo reale che consente *chat* testuali, vocali e video. Il gruppo utilizza tale strumento per comunicare e per interagire velocemente con l'azienda tramite il loro *server* dedicato.

2.1.3.2. Telegram

Applicazione di messaggistica istantanea utilizzata per comunicazioni rapide e brevi all'interno del gruppo.

2.1.3.3. Fogli di Google

Applicazione web per la creazione e la gestione di fogli di calcolo. Il gruppo utilizza tale strumento per tenere traccia delle ore produttive impiegate in ogni *sprint*[g], che saranno successivamente riportate nel documento ufficiale Piano di Progetto.

2.1.3.4. GitHub

Piattaforma per il versionamento del codice e la collaborazione su progetti *software*. Il gruppo utilizza questo strumento per gestire le *repository* relative al prodotto software oggetto del capitolato e per la documentazione, monitorare le attività del progetto, assegnarle ai membri e seguirne lo stato di avanzamento.

2.1.3.5. Typst

Typst[g] è il sistema di impaginazione basato su *markup* che viene adottato dal gruppo per la stesura della documentazione del progetto. Soluzione preferita rispetto a LaTeX per la sua semplicità.

2.1.4. Metriche

Le metriche adottate per il processo di fornitura sono le seguenti:

ID Metrica	Nome
MPC-PV	*Planned Value*
MPC-EV	*Earned Value*
MPC-AC	*Actual Cost*
MPC-CPI	*Cost Performance Index*
MPC-EAC	*Estimated at Completion*
MPC-SV	*Schedule Variance*
MPC-BV	*Budget Variance*
MPC-ETC	*Estimate to Complete*

Tabella 1: Metriche per la fornitura

2.2. Sviluppo

Sempre secondo lo *standard* ISO/IEC 12207:1995 lo scopo del processo di sviluppo è di descrivere le attività e i compiti necessari per creare e mantenere un sistema *software*, garantendo che il prodotto finale soddisfi i requisiti specificati nel contratto.

Di seguito vengono riportate le attività che compongono il processo di sviluppo.

2.2.1. Implementazione del processo

Nella fase di implementazione, il fornitore deve stabilire o selezionare un modello di ciclo di vita appropriato per il progetto. Le attività di sviluppo e i compiti associati vengono mappati su tale modello, considerando che possono essere eseguiti in modo iterativo o ricorsivo. Il fornitore documenta i risultati secondo il processo di documentazione, ne gestisce le modifiche con un sistema di configurazione e risolve eventuali problemi di conformità. Devono essere selezionati strumenti, metodi e linguaggi di programmazione adeguati, e vengono pianificate le azioni necessarie per rispettare gli *standard* di qualità, sicurezza e conformità. Tutto questo è accompagnato dalla creazione di piani documentati e implementabili.

2.2.2. Analisi dei requisiti di sistema

In questa fase, si analizzano le esigenze specifiche del sistema da sviluppare, documentando i requisiti funzionali, di sicurezza, di manutenzione, ergonomici e di interfaccia. Tali requisiti vengono valutati rispetto alla loro tracciabilità, consistenza con le esigenze iniziali, testabilità e fattibilità tecnica. Il risultato è una specifica dettagliata dei requisiti del sistema.

2.2.3. Progettazione architetturale del sistema

Il sistema viene suddiviso in elementi *hardware*, *software* e operazioni manuali, con l'allocazione dei requisiti a ciascun elemento. L'architettura risultante rappresenta un progetto di alto livello che assicura la tracciabilità rispetto ai requisiti e la fattibilità operativa. La documentazione prodotta include la descrizione dell'architettura del sistema e l'assegnazione dei requisiti ai vari componenti.

2.2.4. Analisi dei requisiti software

Per ogni componente *software* individuato, vengono stabiliti e documentati i requisiti dettagliati, tra cui specifiche funzionali, di sicurezza, di ergonomia e requisiti per il database[g]. Tali requisiti sono valutati per verificarne tracciabilità, coerenza interna ed esterna, testabilità e fattibilità progettuale. La fase si conclude con una revisione congiunta per stabilire una *baseline* di requisiti approvati.

2.2.5. Progettazione architetturale del software

I requisiti *software* vengono trasformati in un'architettura che descrive la struttura generale del *software* e identifica i componenti principali. Si definiscono e documentano le interfacce e le prime versioni

della documentazione utente, oltre ai requisiti di *test* preliminari per l'integrazione. La valutazione di questa fase si concentra sulla tracciabilità rispetto ai requisiti, sulla coerenza interna e sull'adeguatezza dei metodi di progettazione utilizzati.

2.2.6. Progettazione dettagliata del software

Si sviluppano progetti dettagliati per ogni componente *software*, che vengono suddivisi in unità più piccole, codificabili e testabili. La documentazione comprende dettagli delle interfacce, del *database* e requisiti specifici per il *testing*. Anche in questa fase, la progettazione è sottoposta a valutazioni che ne verificano tracciabilità, coerenza interna, testabilità e conformità agli *standard* di progettazione.

2.2.7. Codifica e testing del software

I componenti *software* vengono codificati e testati per garantire che soddisfino i requisiti definiti. I risultati dei *test* vengono documentati e la documentazione utente viene aggiornata. Questa fase include anche la preparazione dei requisiti di *test* per l'integrazione *software* e una revisione dell'adeguatezza del codice e dei risultati dei *test*.

2.2.8. Integrazione del software

Le unità e i componenti *software* sono integrate per formare un unico elemento *software*. Durante l'integrazione, vengono eseguiti *test* per verificare che ciascun aggregato soddisfi i requisiti. Si sviluppano piani di *test* e procedure per la qualifica del *software* integrato. L'integrazione è valutata considerando la tracciabilità, la coerenza e la copertura dei requisiti.

2.2.9. Test di qualifica del software

Il *software* integrato viene sottoposto a *test* di qualifica per verificarne la conformità ai requisiti. I risultati di questi *test* vengono documentati. A seguito del completamento con successo, il *software* è pronto per le fasi successive, come l'integrazione di sistema o l'installazione.

2.2.10. Integrazione del sistema

Il *software* viene integrato con componenti *hardware*, operazioni manuali e altri sistemi, se richiesto. Vengono condotti *test* sugli aggregati del sistema per verificarne la conformità ai requisiti. La documentazione prodotta include i risultati dell'integrazione e i *test* effettuati.

2.2.11. Test di qualifica del sistema

L'intero sistema viene testato per verificare la conformità alle specifiche contrattuali e la prontezza per la consegna. I risultati sono documentati e valutati per garantire la copertura dei requisiti e la fattibilità operativa.

2.2.12. Installazione del software

Si sviluppa e implementa un piano per l'installazione del *software* nell'ambiente di destinazione. Si verifica che il *software* funzioni come previsto e si supportano attività di transizione, se necessario. Gli eventi e i risultati dell'installazione sono documentati.

2.2.13. Supporto all'accettazione del software

Il fornitore supporta il cliente nella revisione e nei *test* di accettazione del *software*, che comprendono la valutazione di tutti i risultati ottenuti nelle fasi precedenti. Viene completata la consegna del prodotto *software* e fornite eventuali attività di formazione e supporto iniziale.

2.2.14. Metriche

Le metriche adottate per il processo di sviluppo sono le seguenti:

ID Metrica	Nome
MPC-ISR	Indice di Stabilità dei Requisiti

Tabella 2: Metriche per lo sviluppo

3. Processi di supporto

3.1. Documentazione

Il processo di documentazione è necessario per il tracciamento di tutte le attività relative al progetto, sia dal lato *software* che dal lato organizzativo. Tutti i membri del gruppo si impegnano a rispettare le norme delineate nel capitolo corrente per la stesura di ogni documento.

3.1.1. Preparazione dell'ambiente

3.1.1.1. Tracciamento

Alla fine di ogni riunione si aggiorna il *backlog*[g] con le nuove *issue* (corrispondenti alle decisioni prese durante la riunione). Esse devono specificare in modo esaustivo il compito da svolgere nel titolo, arricchite da una breve descrizione, i verificatori incaricati di visionare il lavoro e dal verbale nel quale è scaturita quella decisione in modo che in futuro sia sempre tracciabile il motivo dell'inserimento di quella *issue*. Devono inoltre essere collegate alla *board* (corrispondente al *backlog*), ad una *milestone* e ad un assegnatario. Una volta create si deve generare, sempre da interfaccia *web*, il *branch*[g] temporaneo nel quale si andrà a lavorare.

3.1.1.2. Lavoro sul documento

Per lavorare su un documento bisogna attenersi ai seguenti passaggi utilizzando il sistema Git[g]:

1. Posizionarsi sulla cartella della *repository* nella macchina locale e scaricare le eventuali modifiche in tutti i *branch* locali eliminando gli eventuali rami che in remoto sono stati eliminati. Se si vuole sincronizzare solo il ramo corrente si deve omettere il *flag* **`--all`**.
   ```
   git pull -p --all
   ```
2. Posizionarsi sul *branch* collegato alla *issue*.
   ```
   git checkout [nome_branch]
   ```
3. Caricare le modifiche effettuate in locale nella *repository* remota.
   ```
   git add .
   git commit -m "[commento del commit]"
   git push
   ```
4. Dall'interfaccia *web* bisogna spostarsi nella sezione ***pull request*** e crearne una nuova assicurandosi di assegnare il verificatore e una *label*. Se si desidera sottoporre le modifiche a un rapido controllo si può creare un ***draft***.
5. In caso il verificatore richieda delle correzioni è sufficiente ripetere i passaggi citati senza questa volta creare una nuova *pull request* perché già esistente. Si ricorda che si può evitare di scaricare le modifiche su tutti i *branch* e che se si è già posizionati sul ramo corretto si può evitare il comando **`git checkout [nome_branch]`**.

Si noti che la modifica di un documento già verificato e approvato deve essere tracciata da una *issue*, quindi i procedimenti per redigere o aggiornare un documento sono gli stessi. Una volta terminato il lavoro si deve aggiornare anche la corrispondenza ruolo-persona nella prima pagina segnando il nuovo redattore, il verificatore e sostituendo il responsabile col compagno che in quel momento sta ricoprendovi il ruolo. In questo modo il responsabile è colui che è al corrente dell'ultima versione, nel caso in cui bisognasse contattarlo per dei chiarimenti.

Se si individuano dei nuovi termini da aggiungere al Glossario si inseriscono secondo le norme decise dal documento corrente (sez. 1.3). Il verificatore deve quindi controllare anche che siano stati inseriti nel modo corretto.

3.1.2. Ciclo di vita

Il ciclo di vita di un documento è composto da 6 fasi:

1. **Creazione e adattamento al *template***: Dopo aver impostato l'ambiente nel modo corretto si può creare il file `.typ` seguendo le norme di nomenclatura. Si importa il *file* `template.typ` della cartella `templates` e si compilano i campi delle funzioni in base al documento che si sta redigendo.
2. **Pianificazione dei paragrafi**: Oltre ai paragrafi dettati dal tipo di documento come specificato dalle norme, si devono aggiungere dei paragrafi che raggruppino gli argomenti trattati.
3. **Stesura del documento**: Il redattore deve stilare il documento seguendo le norme concordate dal gruppo. In caso fosse indeciso su alcuni punti è libero di contattare altri membri del gruppo in modo da scrivere un contenuto il più corretto possibile.
4. **Revisione**: Una volta caricato correttamente il documento viene notificato automaticamente il verificatore il quale deve assicurarsi che non vi siano errori.
5. **Approvazione**: Il documento è approvato dal verificatore e viene contattato il responsabile che deve confermare la pubblicazione nel *branch* principale. Se l'approvazione comporta dei cambiamenti deve essere tracciato nel «Registro delle modifiche», altrimenti viene aumentato il numero della versione delle modifiche più recenti.
6. **Modifiche**: Conseguentemente all'approvazione è possibile dover apportare alcuni accorgimenti. In questo caso si ripetono le operazioni incrementando la versione in conformità con il versionamento adottato.

3.1.3. Struttura dei documenti

Ogni documento prodotto è costituito dalle seguenti sezioni.

3.1.3.1. Intestazione

- **Logo del gruppo**: Presente nella cartella `assets/img`.
- **Titolo del documento**
- **Data**: Data in cui è stato approvato il documento o, nel caso dei verbali, in cui è avvenuta la riunione.
- **Uso**: Interno o Esterno.
- **Destinatari**
- **Responsabile**
- **Redattore**
- **Verificatore**

3.1.3.2. Registro delle modifiche

La seconda pagina è dedicata al registro delle modifiche per i documenti che possiedono un ciclo di vita. Vengono quindi esclusi i verbali e le lettere. Le informazioni sono organizzate in una tabella in ordine cronologico decrescente, quindi con l'ultima modifica effettuata nella prima riga in alto. La tabella riporta i seguenti dati:

- **Versione**
- **Data**: La data nella quale è stata apportata l'ultima modifica nel formato DD/MM/YYYY.
- **Redattori**
- **Verificatori**
- **Descrizione**: Una breve descrizione delle modifiche apportate.

3.1.3.3. Indice

Nella pagina successiva all'ultima occupata dal registro delle modifiche è posto un indice per facilitare la navigazione. Questo viene aggiornato automaticamente con la modifica dei paragrafi nel documento.

3.1.3.4. Corpo dei verbali

Nei verbali è importante non limitarsi a descrivere cosa si è discusso bensì enfatizzare sulle decisioni prese e sulle motivazioni che hanno mosso le stesse. Così facendo si agevola la comprensione dei

colleghi che in futuro dovranno consultare i documenti. I verbali condividono una struttura di base comune generata dal *template*:

- **Informazioni generali**
  - **Luogo e data della riunione**
    - **Luogo**
    - **Data**: Data in cui si è tenuta la riunione nel formato DD/MM/YYYY.
    - **Ora**: Ora di inizio della riunione.
    - **Durata**: Durata della riunione in formato per iscritto (ad esempio 2 ore e 20 minuti).
- **Conclusioni**: Le conclusioni della riunione con le motivazioni alle scelte prese e una previsione delle azioni future.
- **Tabella delle decisioni**: Per una descrizione dettagliata consultare la sezione omonima (sez. 3.1.3.5).

3.1.3.4.1. Verbali Interni

I verbali interni aggiungono le seguenti sezioni:

- **Partecipanti**: L'elenco dei partecipanti, incluso nella sezione «Informazioni generali».
- **Ordine del giorno**

3.1.3.4.2. Verbali Esterni

I verbali esterni aggiungono le seguenti sezioni:

- **Partecipanti interni**: L'elenco dei partecipanti interni, incluso nella sezione Informazioni generali.
- **Partecipanti esterni**: L'elenco dei partecipanti esterni, incluso nella sezione Informazioni generali.
- **Sintesi dell'incontro**
- **Risposte alle domande**

3.1.3.5. Tabella delle decisioni

Una tabella che racchiude le informazioni necessarie per rintracciare gli elementi nel *backlog*. Sono riportati i seguenti dati:

- **ID**: Identificativo della decisione:
  - **[X][ID *issue*]** nel caso fosse rintracciabile mediante *issue* (deve essere fornito un collegamento alla stessa) dove l'ID è scritto con quattro cifre. la X indica una lettera che identifica univocamente la GitHub[g] *repository* alla quale è legata la *issue*:
    - **D**: docs
    - **S**: sweatunipd.github.io
    - **N**: NearYou
- **Assegnatari**: Il membro del gruppo incaricato di svolgere quel compito. Si assegna «Gruppo» nel caso in cui coinvolgesse tutti i membri.
- **Descrizione**: Il titolo della *issue* se esistente, altrimenti una breve descrizione delle azioni da compiere.

3.1.4. Documenti del progetto

Alla conclusione del progetto dovranno essere stati prodotti i seguenti documenti:

- Norme di Progetto
- Piano di Progetto
- Piano di Qualifica
- Analisi dei Requisiti
- Glossario
- Verbali Interni
- Verbali Esterni

3.1.5. Versionamento

Per i documenti soggetti ad un ciclo di vita abbiamo deciso di adottare il *semantic versioning* adattando però lo schema al contesto della documentazione. Sono stati quindi ridefiniti gli indici X.Y.Z come segue:

- **X (*major*)**: Approvazione da parte del responsabile.
- **Y (*minor*)**: Aggiornamento o aggiunta di porzioni di testo consistente.
- **Z (*patch*)**: Aggiustamento dello stile del testo, ortografico e decorativo.

I documenti vengono approvati solo al termine del proprio ciclo di vita. In particolare Piano di Progetto, Norme di Progetto, Piano di Qualifica e Analisi dei Requisiti vengono approvati al termine di ciascuna *milestone*. Nelle fasi intermedie ogni modifica, sottoposta a verifica, incrementa la *minor* o la *patch*.

3.1.6. Nomenclatura

Per la nomenclatura di tutti i file e cartelle si è scelto lo stile *snake case* ad eccezione delle date, scritte in formato YYYY-MM-DD per mantenere l'ordine cronologico, separate dal trattino.

Le cartelle sono divise per *milestone* (candidatura, RTB e PB) ciascuna contenente i verbali interni ed esterni e i documenti del progetto prodotti nel rispettivo periodo. I documenti si dividono quindi nelle seguenti tipologie:

- **documenti soggetti a ciclo di vita**: [nome]_ver[X.Y.Z].
- **documenti non soggetti a ciclo di vita**:
  - **verbali**: [VI/VE]_[YYYY-MM-DD] dove VI e VE indicano rispettivamente verbali interni ed esterni.
  - **lettere**: [nome].

3.1.7. Stile dei titoli

I titoli dei documenti devono seguire le seguenti indicazioni:

- **Verbali Interni**: [NOR] riunione, dove NOR indica il numero ordinale della riunione (ad esempio Prima o Dodicesima).
- **Verbali Esterni**:
  - **SAL fine [NOS] sprint**, dove NOS indica il numero ordinale dello *sprint* (ad esempio primo o dodicesimo) nel caso si trattasse del SAL di fine *sprint*.
  - **SAL intermedio [NOS] sprint**, dove NOS indica il numero ordinale dello *sprint* (ad esempio primo o dodicesimo) nel caso si trattasse di un SAL informativo a metà dello *sprint*.
  - Nel caso non si trattasse di un SAL si deve cercare un titolo breve ma efficace.
- **Documenti**: Per documenti che non sono verbali il titolo deve semplicemente indicare lo scopo del documento (ad esempio Norme di Progetto).

Notare che nei titoli dei documenti e dei paragrafi lo stile rimane quello di *default* di Typst, non vanno aggiunti quindi corsivi o altre personalizzazioni.

3.1.8. Stile del testo

Nei documenti vengono applicate le seguenti regole di stile testuali:

- **Grassetto** col comando `*termine*`:
  - Termini importanti.
- Sottolineato col comando `#link("url")[label]`:
  - ID *issue* nella tabella delle decisioni.
  - Riferimenti contestuali segnando la destinazione con `<dest>` e modificando il comando `#link(<dest>)[label]`.
- **Sottolineato, in grassetto e di colore verde** col comando `#formatLink(label:"label", url:"url")`:

  - Link ipertestuali.
- Sottolineato con apice[g] col comando `#rifGlossario("termine")`:
  - Termini presenti nel Glossario.
- «Virgolette» col comando `"termine"`:
  - Enfatizzare singoli caratteri.
  - Espressioni tecnicamente imprecise (ad esempio «usa e getta»)
- *Corsivo* col comando `_termine_`:
  - Termini in lingua inglese (esclusi se si tratta di nomi di prodotti *software* o simili)
  - Titolo del capitolato *NearYou - Smart custom advertising platform*
- `Monospace` col comando `` `termine` ``:
  - Nomi di file (ad esempio «`norme_di_progetto_verX.Y.Z.typ`»).
  - Nomi di cartelle.
  - Estensioni file.
- Maiuscole:
  - Iniziali di nomi propri.
  - Acronimi (ad esempio «Proof of Concept (PoC[g])»).
  - Prima lettera dei paragrafi se citati nel testo.
  - Nomi dei documenti (ad esempio «Norme di Progetto»).

3.1.9. Riferimenti

3.1.9.1. Contestuali

Per riferirsi ad una sezione del documento per una spiegazione più dettagliata si adotta la dicitura «sez. [I]» dove I indica l'indice della sezione (ad esempio «(sez. 3.2)»). Per un corretto collegamento si veda il paragrafo apposito (sez. 3.1.8).
Nel caso si tratti di un riferimento alla descrizione di una metrica (e quindi solo in questo documento) è sufficiente lasciare come dicitura il codice della metrica stessa (ad esempio «MPC-IG»).

3.1.9.2. Ipertestuali

Per esporre un *link* ad una pagina esterna al documento si utilizza la funzione `#formatLink` (sez. 3.1.8) esplicitando l'interezza dell'*url* anche nella *label*. Unica eccezione di stile per i riferimenti alle *issue* nella tabella delle decisioni (sez. 3.1.8). Ogni riferimento a documenti o pagine esterne va accompagnato dalla data di ultimo accesso o dalla versione.

3.1.10. Elenchi puntati

Una lista è preferibile a un elenco narrativo, da valutare se renderlo numerato o meno a seconda della circostanza. I punti della descrizione nel registro delle modifiche vengono sempre rappresentati in un elenco, eccezione fatta per l'approvazione dei documenti generali. Se le voci dell'elenco sono costituite da delle frasi vengono chiuse da «.», se invece possiedono solo poche parole, che ad esempio indicano il nome di una tecnologia, non viene messo nessun segno di punteggiatura al termine. Le voci dell'elenco della descrizione nel registro delle modifiche non vengono chiuse da punteggiatura.

3.1.11. Formato delle date

Sotto il titolo nella prima pagina dei verbali viene indicata la data per iscritto, quindi nel formato «DD mese YYYY». Le date nei documenti vengono scritte nel formato DD/MM/YYYY, nei nomi dei documenti invece YYYY-MM-DD per mantenere l'ordine cronologico. In tutti i casi si segue la seguente convenzione:
- YYYY: anno con quattro cifre.
- MM: mese con due cifre.
- DD: giorno con due cifre.

3.1.12. Strumenti

Sono stati scelti i seguenti strumenti per redigere e mantenere la documentazione:

- **Typst**: linguaggio per la stesura dei documenti, consigliata l'estensione Tinymist Typst se si usa Visual Studio Code.
- **Draw.io**: sito *web* che offre un'interfaccia grafica per creare i diagrammi utili all'analisi e alla progettazione.
- **GitHub**: servizio di *hosting*[g] di *repository*.

3.1.13. Metriche

Le metriche adottate per la documentazione sono le seguenti:

ID Metrica	Nome
MPC-IG	Indice Gulpease[g]
MPC-CO	Correttezza Ortografica

Tabella 3: Metriche per la documentazione

3.2. Gestione della configurazione

Per gestire la documentazione è stato creata una *repository* contenente tutti i *file* Typst aggiornati e verificati. Gli stessi documenti in formato `.pdf` sono consultabili al sito **https://sweatunipd.github.io** (ultimo accesso 03/03/2025).

L'operazione di *directory listing* di GitHub offre una visione degli artefatti strutturata in cartelle, seguente quindi la composizione della *repository*, all'indirizzo **https://sweatunipd.github.io/docs** (ultimo accesso 03/03/2025).

3.2.1. Repository

Il gruppo utilizza tre *repository* all'interno della propria organizzazione GitHub:

- **docs** (**https://github.com/SWEatUNIPD/docs**, ultimo accesso 03/03/2025) contenente tutta la documentazione del progetto.
- **sweatunipd.github.io** (**https://github.com/SWEatUNIPD/sweatunipd.github.io**, ultimo accesso 03/03/2025) contenente i *file* necessari alla struttura e presentazione del sito *web*.
- **NearYou** (**https://github.com/SWEatUNIPD/NearYou**, ultimo accesso 03/03/2025) contenente il codice sorgente dell'applicativo.

3.2.1.1. Struttura della repository docs

La *repository* è strutturata da un unico *branch* adibito al mantenimento di tutti i documenti Typst verificati. All'occorrenza di svolgere azioni dettate dal *backlog* si crea un *branch* temporaneo che, successivamente alla verifica, viene unito nel ramo principale. La *repository* è presentata dal `README.md`, contiene il *file* `.gitignore` usato da Git per escludere alcuni *file* dai *commit* in *repository* e il *file* `script.js` usato per pubblicare la documentazione nel sito *web*. `test.js` è utilizzato per controllare la presenza dei termini del Glossario all'interno dei documenti, mentre `glossario.json` è una struttura dati che contiene i termini e la loro definizione, usato per la stesura del documento dal *file* `glossario_verX.Y.Z.typ`; è presente inoltre il file `gulpease.js` usato per misurare l'Indice di Gulpease richiesto dalla metrica MPC-IG. Le cartelle sono strutturate nel seguente modo:

- **`.github`**
  - **`workflows`**: contiene i *file* `.yml` per le configurazioni delle GitHub Action.
- **`.vscode`**: contiene le impostazioni di Visual Studio Code specifiche per il progetto.
- **`assets`**
  - **`font`**: contiene i *file* `.ttf` dei *font* usati nei documenti.

- **img**: contiene il logo e le immagini usate nei documenti.
- **use_cases**: cartella dedicata ai diagrammi dei casi d'uso usati nell'Analisi dei Requisiti.

- **candidatura**
  - **verbali**
    - **esterni**: contiene i `.typ` e i `.pdf` dei verbali esterni.
    - **interni**: contiene i `.typ` dei verbali interni.
    - `dichiarazione_impegni_preventivo_costi_ver1.1.0.typ`
    - `lettera_di_presentazione_candidatura.typ`
    - `valutazione_capitolati_ver1.2.0.typ`
- **pb**
  - **verbali**
    - **esterni**: contiene i `.typ` e i `.pdf` dei verbali esterni.
    - **interni**: contiene i `.typ` dei verbali interni.
    - `analisi_dei_requisiti_ver2.0.0.typ`
    - `glossario_ver2.0.0.typ`
    - `manuale_utente_ver1.0.0.typ`
    - `norme_di_progetto_ver2.0.0.typ`
    - `piano_di_progetto_ver2.0.0.typ`
    - `piano_di_qualifica_ver2.0.0.typ`
    - `specifica_tecnica_ver1.0.0.typ`
- **rtb**
  - **verbali**
    - **esterni**: contiene i `.typ` e i `.pdf` dei verbali esterni.
    - **interni**: contiene i `.typ` dei verbali interni.
    - `analisi_dei_requisiti_ver1.0.1.typ`
    - `glossario_ver1.0.0.typ`
    - `lettera_di_presentazione_rtb.typ`
    - `norme_di_progetto_ver1.0.1.typ`
    - `piano_di_progetto_ver1.0.1.typ`
    - `piano_di_qualifica_ver1.0.1.typ`
- **templates**
  - `diario_di_bordo.pptx`
  - `template.typ`
- `.gitignore`
- `README.md`
- *File* `.js` utili per il glossario, per l'Indice di Gulpease e per aggiungere i *file* al sito.
- *File* `.json` per le configurazioni dell'ambiente JavaScript e per il glossario.

3.2.1.2. Struttura della repository sweatunipd.github.io

La *repository* è costituita da un unico *branch* nel quale è caricato tutto il materiale necessario per strutturare il sito *web* e popolarlo dinamicamente con la documentazione della *repository* docs. La *repository* è presentata dal `README.md`, contiene il *file* `.gitignore` necessario per escludere alcuni *file* dai *commit* in *repository*. Le cartelle sono strutturate nel seguente modo:

- **.github**
  - **workflows**: contiene i *file* `.yml` per le configurazioni delle GitHub Action.
- **app**

│ ├─ **fonts**: contiene i *file* `.woff` dei *font* usati nel sito.
│ ├─ `global.css`
│ ├─ `layout.tsx`
│ └─ `page.tsx`
├─ **components**
│ └─ **ui**: contiene i *file* `.tsx` usati per i componenti grafici del sito.
├─ **lib**: contiene funzioni di *utility*[g] di una libreria esterna.
├─ **logo**
│ ├─ `logo.svg`
│ └─ `logo_dark.svg`
├─ **public**
│ └─ `favicon.ico`
├─ `.gitignore`
├─ `README.md`
└─ *File* utili alla configurazione e al trasferimento dati.

3.2.1.3. Struttura della repository NearYou

La *repository* è strutturata da due *branch*: «*main*» e «*dev*». Il *branch* «*main*» contiene la *release* ufficiale più recente e viene aggiornato ad ogni *milestone*, ovvero nel nostro caso RTB e PB, mentre il *branch* «*dev*» è adibito allo sviluppo del codice. Ad ogni necessità di svolgere azioni dettate da una *issue* del *backlog* si crea un *branch* temporaneo che, successivamente alla verifica, viene riunito al ramo «*dev*» secondo il meccanismo delle *pull request*. La *repository* è presentata dal `README.md` il quale fornisce anche le istruzioni su come avviare il sistema tramite Docker[g]. La *repository* contiene anche il *file* `.gitignore` usato per escludere alcuni *file* dai *commit* in *repository*. Le cartelle sono strutturate nel seguente modo:

- **`.github`**
  - **`workflows`**: contiene i *file* `.yml` per le configurazioni delle GitHub Action.
- **`assets`**
  - **`img`**: contiene le immagini usate all'interno dei *file*.
- **`client`**
  - **`src`**: contiene il codice sorgente del simulatore.
  - **`test`**
    - **`unit`**: contiene i *test* di unità.
  - `Dockerfile`
  - *File* di configurazione dell'ambiente TypeScript.
- **`data`**
  - **`grafana`**
    - **`plugins`**: contiene i *plugin* installati per il corretto funzionamento di Grafana[g].
    - **`kui`**: TODO.
- **`job`**
  - **`src`**
    - **`main`**: contiene il codice sorgente del *job* di Flink.
    - **`test`**: contiene i *test* di unità e di integrazione del *job*.
  - `checkstyle.xml`: file di configurazione per l'analisi statica del codice.
  - `pom.xml`: file di configurazione del progetto in Apache Maven[g].
- **`mock`**: contiene i *file* per popolare il *database* con dati fittizi.
- `.gitignore`
- `README.md`
- `compose.yml`
- `create.sql`

3.2.2. Backlog

Il *product backlog* è rappresentato da una GitHub *board* con tre sezioni:

- **Todo**: l'attività non è ancora stata iniziata.
- **In Progress**: l'attività è in corso di svolgimento.
- **Done**: l'attività è stata completata.

3.2.3. Ticketing

Alla fine di ogni riunione interna si aggiorna il *backlog* con le nuove *issue* (corrispondenti alle decisioni prese durante la riunione). Esse devono specificare in modo esaustivo il compito da svolgere nel titolo, arricchite da una breve descrizione e dai riferimenti al verificatore e al verbale nel quale è scaturita quella decisione, in modo che in futuro sia sempre tracciabile il motivo dell'inserimento di quella *issue*. Per una corretta impostazione si chiede di seguire il seguente schema:

| Titolo completo ed esaustivo
| ——————————————————
| Breve descrizione.
|
| Verificatori: @Nickname
|
| Decisione presa nel [VI/VE]_[YYYY-MM-DD]

Devono inoltre essere collegate alla *board* (corrispondente al *backlog*) assicurandosi appaiano nella colonna «Todo», ad una *milestone*, ad una *label* e ad un assegnatario. Una volta create si deve generare, sempre da interfaccia *web*, il *branch* temporaneo nel quale si andrà a lavorare. In questa operazione è sufficiente rinominare il *branch*, collegarci l'assegnatario e assegnarci una *label*.

Si ricorda di aggiornare lo stato di avanzamento della *issue* nel *backlog* alla sezione «In progress» quando la si sta risolvendo. Una volta approvata la *pull request* il sistema chiude in automatico la *issue* posizionandola nella colonna «Done».

3.2.4. Label

Sono state create le seguenti *label* per migliorare l'organizzazione delle *issue* e facilitarne la ricerca tramite filtro:

- **AdR**: Analisi dei Requisiti
- **Automazione**
- **Glossario**:
- **Lettera**: Lettera di candidatura
- **NdP**: Norme di Progetto
- **PdP**: Piano di Progetto
- **PdQ**: Piano di Qualifica
- **Template**
- **Verbale**

3.2.5. GitHub Action

Vengono adoperate delle Action per facilitare il processo di verifica tramite la pubblicazione di un *file* `.zip` nei messaggi delle *pull request* contente i documenti modificati in formato `.pdf` ad ogni *commit*, per automatizzare la pubblicazione dei documenti approvati nel sito del gruppo, per verificare l'Indice di Gulpease dei documenti e per verificare la presenza di termini a glossario non contrassegnati nei documenti.

Per i verbali esterni si è scelto un approccio differente in quanto bisogna aspettare la firma del proponente per presa visione. Non potendo quindi essere pubblicati direttamente nel sito i verbali esterni

vanno compilati localmente e mandati alla controparte. Una volta restituiti firmati vanno caricati manualmente nella stessa cartella dove risiede il codice sorgente e la Action si occuperà di pubblicarli nel sito. Per caricare i documenti firmati bisogna forzare l'operazione col seguente comando:
`git add [file_name] --force`

3.3. Verifica

La verifica è un processo di cruciale importanza che accompagna il *software* lungo tutto il suo ciclo di vita, dalla progettazione fino alla manutenzione. Il suo obiettivo è garantire l'efficienza e la correttezza dei processi e dei loro eventuali prodotti. Un processo di verifica efficace ha come naturale conseguenza un prodotto più stabile, che risulta in un processo di validazione più lineare e prevedibile, meno soggetto ad imprevisti.

3.3.1. Analisi statica

L'analisi statica è un processo di verifica che si svolge senza eseguire il codice ed è perciò applicabile a tutti i prodotti del progetto. Il suo obiettivo è individuare errori e anomalie nel codice sorgente o nel testo prodotto. Le due tecniche principali adottate sono il *walkthrough* e l'*inspection*.

- Il *walkthrough* prevede un'analisi sistematica di tutto il prodotto alla ricerca di irregolarità non meglio specificate.
- L'*inspection*, invece, prevede una ricerca mirata all'interno del codice sorgente o del documento soffermandosi solo su ciò che si sa essere più soggetto ad errori, trasformando quindi il processo di verifica nella risoluzione di una *checklist*.

L'*inspection* è il metodo da preferire in quanto più efficiente, allo stesso tempo è difficilmente implementabile all'inizio per mancanza di esperienza (e quindi dati su quali sono gli errori più comuni).

3.3.2. Analisi dinamica

L'analisi dinamica è un processo di verifica che si svolge eseguendo il codice. Il suo obiettivo è individuare errori e anomalie nel codice sorgente. Lo strumento principale utilizzato è il *test*. I *test* possono essere di vario tipo:

- **Test di unità**: verificano il corretto funzionamento di una singola unità di codice.
- **Test di integrazione**: verificano il corretto funzionamento dell'interazione tra due o più unità di codice.
- **Test di sistema**: verificano il corretto funzionamento del sistema nel suo complesso.
- **Test di regressione**: verificano che le modifiche apportate al codice non abbiano introdotto nuovi errori.
- **Test di accettazione**: verificano che il prodotto soddisfi i requisiti specificati.

3.3.3. Verifica dei documenti

Ogni documento creato o modificato necessita la revisione da uno o più verificatori. Questo processo viene automatizzato il più possibile con l'utilizzo delle *pull request*, il sistema di *ticketing* tramite *issue* e una Action dedicata. Quando possibile è preferibile richiedere la revisione a tutti i verificatori per i documenti di carattere generale perché fondamentali per il corretto svolgimento del progetto.

3.3.3.1. Pull request

Le *pull request* velocizzano e automatizzano la verifica dei documenti. Una volta organizzate nel modo corretto (sez. 3.1.1.2), il verificatore può aggiungere una *review* da interfaccia *web* commentando, chiedendo una modifica o approvando la *pull request*. Una volta soddisfatti i requisiti spetterà al responsabile confermare la nuova versione del documento e unire il *branch* al principale tramite il pulsante «*squash and merge*».

3.3.3.2. Action

Al completamento di ogni *commit* una Action compila i *file* Typst modificati e genera un *file* `.zip` contenente i documenti in formato `.pdf`. Così facendo si è sicuri che il codice è privo di errori sintattici e genera correttamente un prodotto finale e distribuibile. Il verificatore può consultare sia il documento in formato `.pdf` per maggiore leggibilità sia il codice sorgente. Nel caso trovasse degli errori può segnalarli tramite la *pull request*, oppure pubblicare un commento per avviare una discussione riguardo una correzione più complessa.

3.3.3.3. Test

È stato messo a disposizione un *test* il quale controlla che la prima occorrenza dei termini del Glossario venga identificata (sez. 1.3). Viene effettuato automaticamente dalla Action ad ogni *push*, tuttavia è preferibile eseguirlo prima in locale così da efficientare il lavoro. È sufficiente quindi aver installato NodeJS sulla propria macchina ed eseguire il comando **`node test.js`** nella *root* della *repository* locale. Il *test* esamina tutti i documenti e ritorna degli avvisi in caso fallisse.

3.4. Validazione

Il processo di validazione è definibile, secondo lo *standard* ISO/IEC 12207:1995, come il processo di conferma tramite dimostrazioni oggettive che il prodotto soddisfi i requisiti specificati per un sistema. In sostanza, la validazione certifica che il prodotto soddisfi le esigenze del cliente ovvero, più specificamente:

- soddisfi tutti i requisiti concordati;
- sia eseguibile senza problemi nell'ambiente di destinazione.

Si tratta di un processo che si svolge in parallelo alla verifica e che si conclude con l'accettazione del prodotto da parte del cliente. Una volta che il cliente, ovvero nel nostro caso l'azienda proponente, ha accettato il prodotto si può procedere con il rilascio.

Lo strumento principale utilizzato per la validazione è il *test* di accettazione, ovvero l'ultimo *test* che viene eseguito prima del rilascio del prodotto. Tale *test* verrà eseguito in presenza della proponente.

3.5. Risoluzione dei problemi

3.5.1. Documentazione

È possibile incappare in problemi riguardanti il ciclo di redazione e verifica dei documenti. In questi casi è preferibile cercare di risolverli al più presto per conto proprio in maniera da non ostacolare il lavoro dei compagni. Tuttavia se si è insicuri delle procedure da adottare o non si trova una soluzione si può contattare l'amministratore per tornare al più presto operativi. Come ultima alternativa è possibile contattare Klaudio Merja, creatore dell'organizzazione e della *repository* GitHub, il quale è l'unico che può eseguire alcune operazioni forzate poiché gode dei privilegi da amministratore dell'ambiente GitHub.

3.6. Gestione della qualità

3.6.1. Scopo

Il processo di gestione della qualità ha lo scopo di garantire che i prodotti soddisfino i requisiti specificati e che siano, assieme ai processi, conformi agli *standard* di qualità stabiliti. La qualità è un requisito fondamentale per il successo del progetto e deve essere garantita in ogni fase del ciclo di vita del prodotto.

3.6.2. Definizione delle metriche

Lo strumento fondamentale per misurare la qualità di un prodotto o di un processo sono le metriche. Esse permettono di valutare in modo oggettivo il grado di conformità del prodotto o del processo

rispetto agli *standard* di qualità stabiliti. Le metriche vengono utilizzate per tracciare la qualità durante tutto il ciclo di vita del progetto. L'elenco esaustivo di tutte le metriche adottate in questo progetto, e la loro rispettiva descrizione, è consultabile nell'ultima sezione (sez. 6) di questo documento.

3.6.2.1. Identificazione

Le metriche adottate sono identificate da un codice alfabetico, strutturato nel seguente modo:

M[tipo]-[abbreviazione]

dove:

- **tipo**: indica il tipo di metrica, può essere:
  - **PC**: per le metriche riguardanti la qualità di processo.
  - **PD**: per le metriche riguardanti la qualità di prodotto.
- **abbreviazione**: abbreviazione o acronimo del nome della metrica.

3.6.2.2. Struttura

Ciascuna metrica viene descritta tramite seguenti campi:

- **Codice**: codice identificativo della metrica.
- **Nome**: nome della metrica.
- **Descrizione**: breve descrizione della metrica.

e, nei casi in cui sia necessario:

- **Formula**: formula utilizzata per calcolare il valore della metrica.
- **Parametri**: parametri utilizzati nella formula.

3.6.3. Criteri di accettazione

All'interno del Piano di Qualifica sono definiti i criteri di accettazione per le metriche adottate. Questi criteri sono stabiliti in modo da garantire che i prodotti e i servizi soddisfino i requisiti specificati e che siano conformi agli *standard* di qualità stabiliti. Ad ogni metrica è associata una soglia di accettazione e una soglia di ottimalità.

- La soglia di accettazione rappresenta il valore minimo (o massimo) che la metrica deve rispettare per essere considerata accettabile.
- La soglia di ottimalità rappresenta il valore minimo (o massimo) che la metrica deve rispettare per essere considerata ottimale.

3.6.4. Metriche

Le metriche adottate per la gestione della qualità sono le seguenti:

ID Metrica	Nome
MPC-PMS	Percentuale di Metriche Soddisfatte

Tabella 4: Metriche per la gestione della qualità

4. Processi organizzativi

L'ingegneria del *software* è un campo complesso e multidisciplinare che richiede un'attenta pianificazione, una gestione efficace delle risorse e un controllo accurato della qualità. Quindi definire e rispettare processi organizzativi ben organizzati diventa essenziale.

4.1. Gestione dei processi

4.1.1. Scopo

Secondo lo *standard* ISO/IEC 12207:1997 la gestione dei processi mira a stabilire, implementare e migliorare i processi che il gruppo deve assicurarsi vengano svolti per garantire la massima resa dei processi di sviluppo.

4.1.2. Descrizione

Le attività di gestione dei processi sono:

1. Definizione dei processi:
   - Identificare e documentare i processi.
   - Stabilire linee guida e procedure per l'esecuzione di ciascun processo.
2. Pianificazione e monitoraggio:
   - Elaborare piani dettagliati per l'esecuzione dei processi.
   - Monitorare costantemente l'avanzamento, l'efficacia e la conformità ai requisiti.
   - Stimare i tempi, le risorse ed i costi.
3. Esecuzione, revisione e valutazione:
   - Monitorare l'avanzamento dei processi per identificare aree problematiche o di miglioramento.
   - Con l'avanzamento implementare azioni correttive e preventive per ottimizzare i processi.
4. Formazione e competenze:
   - Assicurare che il personale coinvolto in un processo sia adeguatamente formato rispetto al dominio d'uso di esso.
5. Chiusura.

4.1.3. Pianificazione

4.1.3.1. Descrizione

Come stabilito dallo *standard* ISO/IEC 12207:1997 il responsabile è responsabile della preparazione dei piani per l'esecuzione di tutte le attività relative alla pianificazione del periodo di carica. Ogni attività dovrà avere associata una descrizione, il personale incaricato di gestire i processi di essa e una scadenza da rispettare.

Il responsabile di ciascuno *sprint* ha il compito di redigere questa pianificazione all'interno del documento Piano di Progetto, che riporterà le attività da svolgere in quel periodo.

4.1.3.2. Scopo

Gli scopi della pianificazione sono avere un piano reale di quelli che sono gli obiettivi realistici dello *sprint*, per poi poter eseguire una retrospettiva, e suddividere tutte le attività necessarie per perseguire questi obiettivi nelle risorse del *team*, in modo da garantire ordine e chiarezza nella divisione dei compiti.

4.1.3.3. Ruoli

I ruoli descritti sotto verranno assegnati ad ogni *sprint* in base alle esigenze della fase del progetto, con una pianificazione in accordo con il documento di Dichiarazione Impegni e Preventivo Costi presentato in fase di candidatura.

4.1.3.3.1. Responsabile

Il responsabile si occuperà dell'allocazione delle risorse e della pianificazione generale e si assicurerà che il progetto proceda in modo efficiente e soddisfi gli obiettivi nel rispetto delle scadenze. Sarà inoltre la figura incaricata a rappresentare i *team* nelle varie interazioni con i proponenti e i committenti. I suoi compiti sono:

- Approvare la documentazione.
- Gestire la pianificazione e la retrospettiva dello *sprint* in cui è incaricato.
- Studiare e gestire l'analisi dei rischi.
- Rappresentare il gruppo nelle comunicazioni con l'esterno.

4.1.3.3.2. Amministratore

L'amministratore garantirà che l'ambiente e l'infrastruttura necessari per lo sviluppo e l'esecuzione del progetto siano robusti, sicuri ed affidabili e che ciascun membro del *team* sappia come usarli senza incorrere in alcun tipo di problema. I suoi compiti sono:

- Sostituire il responsabile in caso di emergenze.
- Migliorare l'ambiente di lavoro automatizzando il più possibile i processi.
- Redigere e mantenere la documentazione.
- Gestire le infrastrutture e i servizi.

4.1.3.3.3. Analista

L'analista definisce quali dovranno essere le funzionalità del *software*, in accordo con le necessità del cliente, producendo il documento di Analisi dei Requisiti. Ha il compito di:

- Studiare il problema descritto dal proponente e il relativo dominio d'uso.
- Raccogliere e studiare i bisogni dei committenti.
- Redigere il documento Analisi dei Requisiti studiando tutti i casi d'uso.

4.1.3.3.4. Progettista

Il progettista si occuperà del *design* del *software*: partendo dai problemi identificati dall'Analisi dei Requisiti elabora una possibile soluzione, che successivamente dovrà essere implementata dai programmatori, fungendo quindi da «elaborazione intermedia» tra le due parti. Ha il compito di:

- Progettare un prodotto economico e manutenibile partendo dal lavoro degli analisti.
- Garantire che vengano rispettati i principi tecnici dell'ingegneria del *software*, come un livello di accoppiamento più basso possibile.

4.1.3.3.5. Verificatore

Il verificatore dovrà garantire che il *software* mantenga adeguati *standard* di qualità garantendone, attraverso *test* e controlli, l'affidabilità e la robustezza. Ha il compito di:

- Verificare la correttezza di contenuto e stilistica dei documenti con le regole definite nelle Norme di Progetto.
- Verificare che il codice prodotto sia privo di *bug* e in linea con le linee guida dei progettisti.

4.1.3.3.6. Programmatore

Il programmatore si occuperà di tradurre le specifiche e i requisiti del progetto nel sistema funzionante richiesto dal proponente. Ha il compito di:

- Implementare il prodotto rispetto alle linee guida dei progettisti.
- Scrivere codice mantenibile, che rispetti le Norme di Progetto.
- Produrre *test* per la verifica e validazione del codice.
- Redigere il manuale utente.

4.1.3.4. Ticketing

Il gruppo adotta l'**Issue Tracking System** (ITS) interno di GitHub. Esso permette una gestione semplice e chiara dei compiti da svolgere grazie alla *project board* e alle *issues*, che possono essere chiuse automaticamente a lavoro svolto.

Ad ogni riunione di inizio *sprint* il responsabile crea le *issues* e le assegna ai vari membri del gruppo. Lo stato di avanzamento delle *issues* è consultabile all'interno della *project board*.

Le *issues* sono create dal responsabile e sono composte da:

- **ID**: identifica in modo univoco la *issue* e viene riportato nel *backlog* e nella tabella delle decisioni del verbale in cui viene presa la decisione.
- **Titolo**: identifica il compito da svolgere.
- **Descrizione**: descrizione breve dell'attività da svolgere, riferimento al verbale in cui è stata presa tale decisione per avere più informazioni riguardo ad essa e i membri del gruppo che dovranno verificare l'attività svolta.
- **Assegnatario**: i componenti del gruppo incaricati a svolgere l'attività.
- **Milestone**: la *release* in cui deve essere pubblicato il risultato dell'attività.
- **Etichetta**: la categoria a cui appartiene quella attività.
- **Stato**: avanzamento del *task*.
- **Data di inizio attività**: momento di presa in carico dell'attività, fondamentale per aggiornare il diagramma di Gantt[g].
- **Data di fine attività**: il termine indicato per portare a termine l'attività, fondamentale per aggiornare il diagramma di Gantt.

Ogni qualvolta ci sia la necessità di portare a termine un compito è necessario seguire la seguente procedura:

1. Il responsabile, dopo aver concordato in una riunione il da farsi, crea una nuova *issue* con stato «To Do» su GitHub e la assegna.
2. All'inizio del lavoro di produzione l'assegnatario cambia lo stato della *issue*, passando da «To Do» ad «In Progress», creando un nuovo *branch* apposito staccato dal ramo principale per eseguire l'attività.
3. Contestualmente l'assegnatario apre una *pull request* su GitHub e assegna i verificatori per quell'attività.
4. Il verificatore controlla il lavoro svolto:
   - Se la verifica ha esito positivo:
     1. Il verificatore approva su GitHub la *pull request* e chiede al responsabile di effettuare il *merge* nel ramo principale .
     2. La *issue* viene marcata «Done» su GitHub automaticamente.
   - Se la verifica ha esito negativo:
     1. Il verificatore rilascia una lista di cambiamenti suggeriti nella *pull request*.
     2. L'incaricato apporta le modifiche suggerite e si torna al punto 4.

4.1.4. Coordinamento

Il coordinamento è l'attività che riguarda la gestione delle comunicazioni e degli incontri tra i membri del *team*, proponenti e committenti del progetto. L'efficienza del progetto e il coinvolgimento di tutte le parti interessate dipendono dal coordinamento.

La gestione della comunicazione interna ed esterna, la conduzione delle riunioni e la definizione di comportamenti comuni per i membri del *team* sono esempi di attività di coordinamento.

4.1.4.1. Comunicazioni

4.1.4.1.1. Comunicazioni interne

Le comunicazioni interne vengono gestite con due strumenti separati: Telegram per le comunicazioni rapide da dispositivi *mobile* e Discord per le riunioni e le chiamate tra i membri del *team*.

4.1.4.1.2. Comunicazioni esterne

Il responsabile del progetto sarà incaricato di gestire il dialogo con l'esterno attraverso l'indirizzo *e-mail* **sweat.unipd@gmail.com**. Per quanto riguarda il mezzo di comunicazione per i SAL con i proponenti lo strumento utilizzato è *Google Meet* che rende le videochiamate accessibili a tutti.

4.1.4.2. Riunioni

Al fine di garantire l'efficienza delle riunioni il responsabile corrente avrà il compito di interloquire con i proponenti per riepilogare i punti principali del periodo trascorso e esprimere dubbi e incertezze emerse dal gruppo.

4.1.4.2.1. Riunioni Interne

Le riunioni interne sono programmate di comune accordo tra i membri del gruppo. In caso di necessità è possibile richiedere riunioni straordinarie durante la settimana tramite il canale dedicato su Telegram con data e orario stabiliti attraverso un sondaggio. Tutte le riunioni *online* si svolgeranno nel canale Discord appositamente designato. Nelle riunioni interne l'obiettivo finale è quello di risolvere le mancanze segnalate dall'ordine del giorno, per cui nella fase finale della riunione il responsabile, dopo aver discusso con il gruppo, pianifica le attività da svolgere e crea rispettivamente le *issue* come da procedura sopra.

4.1.4.2.2. Riunioni Esterne

Le riunioni esterne coinvolgono i membri del gruppo e il proponente.

Per le riunioni con il proponente viene utilizzata la piattaforma Google Meet e il *link* per accedere alla chiamata viene comunicato al *team* di volta in volta. In accordo con l'azienda proponente gli incontri di fine *sprint* avvengono ogni due settimane, in corrispondenza della cadenza degli *sprint*, mentre a metà di ogni *sprint* si svolgerà un incontro di allineamento più breve per un confronto e degli aggiustamenti.

4.1.4.2.3. Compiti del responsabile

- Esporre i punti all'ordine del giorno.
- Fare il punto su quanto fatto e sul rispetto degli obiettivi posti.
- Pianificare le prossime attività da svolgere.
- Assegnare i *task* ai membri del gruppo.

4.1.4.3. Verbali

4.1.4.3.1. Verbali Interni

Al fine di tracciare le discussioni svolte, le decisioni prese dal gruppo e soprattutto le motivazioni dietro ad esse, al termine di ogni incontro interno viene aperta una *issue* su Github per la preparazione, la verifica e l'approvazione del verbale. Il compito di redigere il verbale, seguendo il formato indicato nella sez. 3.1 di questo documento, è affidato al responsabile il quale deve assicurarsi di includere tutte le informazioni rilevanti discusse.

4.1.4.3.2. Verbali Esterni

Per quanto riguarda le riunioni con i proponenti valgono le stesse regole di quelle interne con il passaggio aggiuntivo, di cui si occupa il responsabile, di richiedere l'approvazione del proponente tramite la sua firma sul verbale una volta redatto e verificato.

4.1.5. Metriche

ID Metrica	Nome
MPC-ET	Efficienza Temporale

Tabella 5: Metriche per la gestione dei processi

4.2. Miglioramento

4.2.1. Descrizione

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

- Analisi: identificare le aree di miglioramento dei processi.
- Miglioramento: implementare le modifiche necessarie per migliorare i processi di sviluppo del *software*.

4.2.1.1. Analisi dei processi

Innanzitutto occorre stabilire una serie di processi organizzativi per l'intero ciclo di vita del *software* applicabili alle varie attività di progetto. Quest'ultimi devono essere documentati e va implementato un meccanismo di controllo per sviluppare, monitorare e migliorare i processi stessi. Per garantire efficacia e continuo miglioramento dei processi alla fine di ogni *sprint* il gruppo effettuerà una retrospettiva del lavoro svolto nel periodo trascorso che verrà verbalizzata dal responsabile nel documento Piano di Progetto.

4.2.1.2. Miglioramento dei processi

Una volta identificati i potenziali miglioramenti questi vanno effettivamente implementati secondo le corrette e adeguate metodologie. Nella retrospettiva seguente all'applicazione di queste migliorie verrà effettuato un altro *feedback*[g] a riguardo per progredire ciclicamente.

4.2.2. Strumenti

Lo strumento principale per misurare l'efficacia delle modifiche apportate ad un processo e per identificare eventuali aree di miglioramento sono le metriche, ognuna delle quali analizzerà un aspetto chiave del processo stesso. Il loro andamento è consultabile nel Cruscotto della Qualità.

4.3. Formazione

4.3.1. Descrizione

Per iniziare a formare i membri del gruppo, è necessario prima comprendere completamente il dominio del problema. È necessario quindi comprendere quali argomenti sono necessari per essere approfonditi e quali abilità sono necessarie per i vari processi.

Si deve quindi passare all'individuazione del materiale di formazione che aumenterà nel tempo poiché il nostro livello di conoscenza e comprensione del problema dovrà aumentare man mano che il progetto avanza.

Infine tutti i membri del gruppo devono imparare da dove e cosa studiare e devono anche aggiornarsi individualmente, se possibile, o con l'aiuto di altri membri più esperti.

4.3.2. Strumenti

Al fine di agevolare il processo di formazione ogni membro del gruppo incaricato in un determinato *sprint* di approfondire un determinato argomento dovrà essere in grado di tramandare la propria conoscenza agli altri. Per aiutarci nel processo di apprendimento l'azienda Sync Lab S.r.l. ci fornisce,

durante lo svolgimento del progetto, materiali quali video e risorse utili alle tecnologie di dominio d'uso del nostro capitolato, nonché delle sedute di *deep dive* tecnologico concordate a calendario.

5. Standard per la qualità

Per la valutazione della qualità del *software* prodotto il gruppo si prefigge di adottare le linee guida dello *standard* ISO/IEC 9126.

5.1. Standard ISO/IEC 9126

L'ISO/IEC 9126 è uno *standard* internazionale creato per la valutazione della qualità del *software*. Definisce un modello di qualità del *software* in termini di sei caratteristiche generali e venticinque sotto-caratteristiche. Questo *standard* è stato sostituito dall'ISO/IEC 25010, ma rimane comunque un riferimento importante per la valutazione della qualità del *software*.

Di seguito forniamo una panoramica delle caratteristiche reputate rilevanti per il nostro progetto del modello di qualità descritto dallo *standard* ISO/IEC 9126.

5.1.1. Funzionalità

La funzionalità è la capacità del prodotto *software* di fornire funzioni che soddisfano le esigenze esplicite e implicite necessarie per operare.
Questa caratteristica è composta dalle seguenti sotto-caratteristiche:

- **Appropriatezza**: capacità del prodotto di fornire un adeguato insieme di funzioni per consentire all'utente di perseguire i suoi specifici compiti e obiettivi.
- **Accuratezza**: capacità del prodotto di fornire i risultati o gli effetti attesi.
- **Interoperabilità**: capacità del prodotto di interagire con uno o più sistemi specificati.
- **Conformità**: capacità del prodotto di aderire a *standard*, convenzioni e regolamentazioni rilevanti per il settore operativo a cui vengono applicate.
- **Sicurezza**: capacità del prodotto di proteggere i dati e le funzioni da accessi non autorizzati.

5.1.2. Affidabilità

L'affidabilità è la capacità del prodotto di mantenere un certo livello di prestazioni in condizioni specificate per un periodo di tempo specificato.
Le sue sotto-caratteristiche sono:

- **Maturità**: capacità del prodotto di evitare errori, malfunzionamenti o arresti inaspettati.
- **Tolleranza agli errori**: capacità del prodotto di mantenere un livello prestabilito di prestazioni in caso di errori.
- **Recuperabilità**: capacità del prodotto di ripristinare il livello di prestazioni e i dati in caso di malfunzionamenti.
- **Aderenza**: capacità del prodotto di aderire a *standard*, convenzioni e regolamentazioni riguardanti l'affidabilità.

5.1.3. Efficienza

L'efficienza è la capacità del prodotto di fornire prestazioni appropriate rispetto alla quantità di risorse utilizzate.
Le sue sotto-caratteristiche sono:

- **Comportamento rispetto al tempo**: capacità del prodotto di fornire, sotto certe condizioni, adeguati tempi di risposta, elaborazione e velocità di attraversamento.
- **Utilizzo delle risorse**: capacità del prodotto di utilizzare adeguatamente risorse come memoria, CPU e spazio su disco.
- **Conformità**: capacità del prodotto di aderire a *standard* e specifiche per l'efficienza.

5.1.4. Usabilità

L'usabilità è la capacità del prodotto di essere compreso, appreso, utilizzato e attraente per l'utente.
Le sue sotto-caratteristiche sono:

- **Comprensibilità**: capacità del prodotto di rendere i suoi concetti facilmente comprensibili all'utente, permettendogli di valutare se il prodotto è adatto alle sue esigenze.
- **Apprendibilità**: capacità del prodotto essere facilmente apprendibile per gli utenti che non ne conoscono già il funzionamento.
- **Operabilità**: capacità del prodotto di essere utilizzato dagli utenti per i propri scopi e controllandone l'uso.
- **Attrattività**: capacità del prodotto di essere piacevole da utilizzare per l'utente.
- **Conformità**: capacità del prodotto di aderire a *standard* o convenzioni per l'usabilità.

5.1.5. Manutenibilità

La manutenibilità è la capacità del prodotto di essere modificato, includendo correzioni, miglioramenti o adattamenti.
Le sue sotto-caratteristiche sono:

- **Analizzabilità**: capacità del prodotto di essere facilmente analizzato per identificare difetti o cause di malfunzionamenti.
- **Modificabilità**: capacità del prodotto di essere facilmente modificato (sostituendo componenti).
- **Stabilità**: capacità del prodotto di evitare effetti indesiderati derivanti da modifiche.
- **Testabilità**: capacità del prodotto di essere facilmente testato così da validare le modifiche apportate.

6. Metriche per la qualità

6.1. Metriche per la qualità di processo

6.1.1. Processi primari

6.1.1.1. Fornitura

- **MPC-PV**:
  - **Nome**: *Planned Value*
  - **Descrizione**: Indica il valore che si prevede di aver prodotto fino a quel momento.
  - **Formula**:

$$\mathrm{PV} = \%\mathrm{LSP} * \mathrm{BAC}$$

  - **Parametri**:
    - **%LSP**: Percentuale di Lavoro Svolto secondo la Pianificazione (ore pianificate rispetto alle ore totali disponibili).
    - **BAC**: *Budget at Completion*, ovvero il costo totale del progetto, stabilito in fase di candidatura.
- **MPC-EV**:
  - **Nome**: *Earned Value*
  - **Descrizione**: Indica il valore del lavoro effettivamente svolto fino a quel momento.
  - **Formula**:

$$\mathrm{EV} = \%\mathrm{LSE} * \mathrm{BAC}$$

  - **Parametri**:
    - **%LSE**: Percentuale di Lavoro Svolto Effettivamente (ore consumate rispetto al totale disponibile).
    - **BAC**: *Budget at Completion.*
- **MPC-AC**:

  - **Nome**: *Actual Cost*
  - **Descrizione**: Indica i costi effettivi sostenuti dall'inizio del progetto fino a quel momento. Ricavabile in qualsiasi momento consultando il Piano di Progetto.
- **MPC-CPI**:
  - **Nome**: *Cost Performance Index*
  - **Descrizione**: Indica il rapporto tra il valore guadagnato e il costo effettivo. Più grande il suo valore, maggiore sarà l'efficienza.
  - **Formula**:

$$\mathrm{CPI} = \frac{\mathrm{EV}}{\mathrm{AC}}$$

  - **Parametri**:
    - **EV**: *Earned Value*, ovvero il valore guadagnato.
    - **AC**: *Actual Cost*, ovvero il costo effettivo.
- **MPC-EAC**:
  - **Nome**: *Estimated at Completion*
  - **Descrizione**: Indica il costo stimato per terminare il progetto se si mantenesse l'attuale efficienza nell'utilizzo delle risorse.
  - **Formula**:

$$\mathrm{EAC} = \frac{\mathrm{BAC}}{\mathrm{CPI}}$$

  - **Parametri**:
    - **BAC**: *Budget at Completion.*
    - **CPI**: *Cost Performance Index.*
- **MPC-SV**:
  - **Nome**: *Schedule Variance*
  - **Descrizione**: Indica se il progetto è in anticipo (valore positivo), in pari (valore zero) o in ritardo (valore negativo) rispetto alla pianificazione.
  - **Formula**:

$$\mathrm{SV} = \mathrm{EV} - \mathrm{PV}$$

  - **Parametri**:
    - **EV**: *Earned Value.*
    - **PV**: *Planned Value.*
- **MPC-BV**:
  - **Nome**: *Budget Variance*
  - **Descrizione**: Indica se i costi finora sostenuti per il progetto sono meno (valore positivo) o più (valore negativo) del previsto.
  - **Formula**:

$$\mathrm{BV} = \mathrm{PV} - \mathrm{AC}$$

  - **Parametri**:
    - **PV**: *Planned Value.*
    - **AC**: *Actual Cost.*
- **MPC-ETC**:
  - **Nome**: *Estimate to Complete*

- **Descrizione**: Costo stimato per poter completare il progetto allo stato attuale.
- **Formula**:

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

- **Parametri**:
  - **EAC**: *Estimated at Completion.*
  - **AC**: *Actual Cost.*

6.1.1.2. Sviluppo

- **MPC-ISR**:
  - **Nome**: Indice di Stabilità dei Requisiti
  - **Descrizione**: Indice che misura la variazione dei requisiti nel corso del tempo.
  - **Formula**:

$$\mathrm{ISR} = 100 - \left(\frac{\mathrm{RM} + \mathrm{RC} + \mathrm{RA}}{\mathrm{RT}}\right) * 100$$

  - **Parametri**:
    - **RM**: Numero di Requisiti Modificati.
    - **RC**: Numero di Requisiti Cancellati.
    - **RA**: Numero di Requisiti Aggiunti.
    - **RT**: Numero Totale di Requisiti inizialmente previsti.

6.1.2. Processi di supporto

6.1.2.1. Documentazione

- **MPC-IG**:
  - **Nome**: Indice «Gulpease»
  - **Descrizione**: Indica il livello di leggibilità di un testo:
    - Inferiore a 80: difficile da leggere per chi ha la licenza elementare.
    - Inferiore a 60: difficile da leggere per chi ha la licenza media.
    - Inferiore a 40: difficile da leggere per chi ha la licenza superiore.
  - **Formula**:

$$\mathrm{IG} = 89 + \frac{300 * \mathrm{NDF} - 10 * \mathrm{NDL}}{\mathrm{NDP}}$$

  - **Parametri**:
    - **NDF**: Numero di Frasi presenti nel testo.
    - **NDL**: Numero di Lettere presenti nel testo.
    - **NDP**: Numero di Parole presenti nel testo.
- **MPC-CO**:
  - **Nome**: Correttezza Ortografica
  - **Descrizione**: Indica il numero di errori ortografici trovati nel testo.

6.1.2.2. Gestione della qualità

- **MPC-PMS**:
  - **Nome**: Percentuale di Metriche Soddisfatte
  - **Descrizione**: Indica la percentuale di metriche che risultano soddisfare gli obiettivi minimi di qualità previsti dal Piano di Qualifica.
  - **Formula**:

$$\text{PMS} = \left(\frac{\text{MS}}{\text{MT}}\right) * 100$$

- **Parametri**:
  - **MS**: Numero di Metriche Soddisfatte.
  - **MT**: Numero di Metriche Totali.

6.1.3. Processi organizzativi

6.1.3.1. Gestione dei processi

- **MPC-ET**:
  - **Nome**: Efficienza Temporale
  - **Descrizione**: Indica la percentuale di tempo disponibile che il *team* è riuscito a impiegare svolgendo attività produttive, ossia quelle che contribuiscono direttamente al raggiungimento degli obiettivi del progetto.
  - **Formula**:

$$\text{ET} = \left(\frac{\text{OO}}{\text{OP}}\right) * 100$$

  - **Parametri**:
    - **OO**: Ore di Orologio (tempo totale trascorso).
    - **OP**: Ore Produttive (tempo effettivamente dedicato ad attività produttive).

6.2. Metriche per la qualità di prodotto

6.2.1. Funzionalità

- **MPD-ROS**
  - **Nome**: Requisiti Obbligatori Soddisfatti
  - **Descrizione**: Misura la percentuale di requisiti obbligatori soddisfatti dal prodotto.
  - **Formula**:

$$\%\text{ROS} = \frac{\text{NROS}}{\text{NTRO}} * 100$$

  - **Parametri**:
    - **NROS**: Numero di Requisiti Obbligatori Soddisfatti.
    - **NTRO**: Numero Totale di Requisiti Obbligatori.
- **MPD-RDS**
  - **Nome**: Requisiti Desiderabili Soddisfatti
  - **Descrizione**: Misura la percentuale di requisiti desiderabili soddisfatti dal prodotto.
  - **Formula**:

$$\%\text{RDS} = \frac{\text{NRDS}}{\text{NTRD}} * 100$$

  - **Parametri**:
    - **NRDS**: Numero di Requisiti Desiderabili Soddisfatti.
    - **NTRD**: Numero Totale di Requisiti Desiderabili.
- **MPD-RFS**
  - **Nome**: Requisiti Facoltativi Soddisfatti
  - **Descrizione**: Misura la percentuale di requisiti facoltativi soddisfatti dal prodotto.

- **Formula**:

$$\%\text{RFS} = \frac{\text{NRFS}}{\text{NTRF}} * 100$$

- **Parametri**:
  - **NRFS**: Numero di Requisiti Facoltativi Soddisfatti.
  - **NTRF**: Numero Totale di Requisiti Facoltativi.

6.2.2. Affidabilità

- **MPD-CC**
  - **Nome**: Code Coverage[g]
  - **Descrizione**: Misura la percentuale di codice eseguita durante i *test*. Per questo progetto è richiesta una copertura pari o superiore all'80%.
- **MPD-BC**
  - **Nome**: Branch Coverage
  - **Descrizione**: Calcola la percentuale di rami decisionali del codice eseguiti durante i *test*.
- **MPD-PTCP**
  - **Nome**: Passed Test Cases Percentage
  - **Descrizione**: Misura la percentuale di *test* superati rispetto ai test eseguiti.

6.2.3. Efficienza

- **MPD-UR**
  - **Nome**: Utilizzo di Risorse
  - **Descrizione**: Misura l'efficienza del sistema in termini di utilizzo delle risorse *hardware*, come CPU, memoria e altre risorse di sistema.

6.2.4. Usabilità

- **MPD-FU**
  - **Nome**: Facilità di Utilizzo
  - **Descrizione**: Misura il numero di errori commessi dagli utenti durante l'interazione. Un valore minimo indica un'interfaccia intuitiva.
- **MPD-TA**
  - **Nome**: Tempo di Apprendimento
  - **Descrizione**: Valuta il tempo necessario a un utente per imparare a utilizzare il *software*.

6.2.5. Manutenibilità

- **MPD-CCM**
  - **Nome**: Complessità Ciclomatica per Metodo
  - **Descrizione**: Valuta la complessità per metodo del codice sorgente attraverso la misurazione del numero di cammini indipendenti attraverso il grafo di controllo del flusso.
  - **Formula**:

$$\text{CC} = \text{e} - \text{n} + 2$$

- **MPD-CS**
  - **Nome**: Code Smell
  - **Descrizione**: Rileva potenziali problemi di progettazione o codice che potrebbero richiedere manutenzione.