Norme di progetto

Gruppo	Alt+F4
Data	4 Giugno 2025
Versione	v2.0

Registro modifiche

Versione	Data	Autore/i	Verificatore/i	Descrizione
v2.0	4 Giugno 2025		Marko Peric	Release
v1.14	30 Marzo 2025	Eghosa Matteo Igbinedion Osamwonyi	Marko Peric	Scrittura sezione: Attività di Codifica
v1.13	30 Marzo 2025	Enrico Bianchi	Marko Peric	Scrittura sezione: Processo di Miglioramento
v1.12	30 Marzo 2025	Enrico Bianchi	Guirong Lan	Aggiunte nuove metriche di prodotto al Processo di accertamento qualità
v1.11	30 Marzo 2025	Marko Peric	Enrico Bianchi	Aggiunta sezione Attività di progettazione
v1.10	30 Marzo 2025	Guirong Lan	Enrico Bianchi	Correzioni al Processo di Verifica
v1.9	27 Marzo 2025	Enrico Bianchi	Francesco Savio	Aggiunta sezione testing di codice
v1.8	27 Marzo 2025	Marko Peric	Pedro Leoni	Stesura sezione al Processo di validazione
v1.7	27 Marzo 2025	Marko Peric	Pedro Leoni	Stesura sezione Risoluzione dei problemi
v1.6	27 Marzo 2025	Francesco Savio	Pedro Leoni	Correzione processo Gestione della configurazione, stesura sezione Strumenti
v1.5	26 Marzo 2025	Guirong Lan	Pedro Leoni	Correzioni al Processo Documentazione

v1.4	26 Marzo 2025	Pedro Leoni	Marko Peric	Correzioni al Analisi dei requisiti
v1.3	25 Marzo 2025	Enrico Bianchi	Pedro Leoni	Correzioni al Processo di Gestione
v1.2	25 Marzo 2025	Marko Peric	Pedro Leoni	Modifica sezione Processo di fornitura
v1.1	25 Marzo 2025	Enrico Bianchi	Pedro Leoni	Correzioni al Processo di accertamento qualità
v1.0	17 Marzo 2025		Enrico Bianchi	Release
v0.15	16 Marzo 2025	Enrico Bianchi	Francesco Savio	Verifica generale e correzioni
v0.14	27 Febbraio 2025	Marko Peric	Francesco Savio	Scrittura sezioni: StarUML, Requisiti Software
v0.13	9 Gennaio 2025	Enrico Bianchi, Matteo Eghosa	Marko Peric	Scrittura sezione: Processo di accertamento qualità
v0.12	8 Gennaio 2025	Pedro Leoni	Marko Peric	Modifica sezione: Casi d'uso, Comandi personalizzati, Ambienti personalizzati
v0.11	7 Gennaio 2025	Marko Peric	Enrico Bianchi	Modifica sezione: Creazione delle tabelle, Aggiunta sezione: Creazione dei grafici
v0.10	31 Dicembre 2024	Marko Peric	Francesco Savio	Modifica sezioni: Github action controllo delle ore di lavoro, Github action del controllo ortografico dei documenti LaTeX

v0.9	26 Dicembre 2024	Marko Peric	Enrico Bianchi	Modifica sezione: Documentazione, introdotta descrizione struttura documento Piano di Progetto
v0.8	23 dicembre 2024	Francesco Savio	Marko Peric	Modifica sezione: Processi di supporto, Processo di infrastruttura
v0.7	22 dicembre 2024	Enrico Bianchi, Guirong Lan	Marko Peric	Scrittura sezione: Processo di Verifica
v0.6	21 dicembre 2024	Guirong Lan	Marko Peric	Scrittura sezione: Descrizione Processi Primari, Processo di acquisizione, Processo di fornitura
v0.5	11 dicembre 2024	Pedro Leoni	Francesco Savio, Enrico Bianchi	Scrittura sezione: Documentazione
v0.4	11 dicembre 2024	Pedro Leoni	Francesco Savio, Enrico Bianchi	Scrittura sezione: Gestione della configurazione
v0.3	11 dicembre 2024	Pedro Leoni, Francesco Savio	Francesco Savio, Enrico Bianchi	Scrittura sezione: Processo di infrastruttura
v0.2	11 dicembre 2024	Pedro Leoni, Enrico Bianchi	Francesco Savio, Enrico Bianchi	Scrittura sezione: Gestione di progetto
v0.1	11 dicembre 2024	Pedro Leoni, Francesco Savio, Matteo Eghosa	Francesco Savio, Enrico Bianchi	Scrittura sezione: Processo di sviluppo

Indice

1 Processi primari

I processi primari definiti dalla norma *ISO/IEC 12207:1996*$_G$ sono essenziali per la realizzazione di un progetto e sono strettamente legati all'acquisizione, alla realizzazione, alla gestione, all'operazione e al ritiro del software durante l'intero *ciclo di vita*$_G$. Un processo primario consiste in un insieme di attività fondamentali e interconnesse, e variano a seconda della tipologia di processo. Questi processi sono cruciali nelle fasi operative del *ciclo di vita*$_G$ del software e sono determinanti per garantire una produzione, manutenzione e dismissione in modo efficace. La norma distingue cinque principali processi primari, ognuno dei quali comprende attività chiave per lo sviluppo e la gestione del software, dalla sua concezione fino al suo ritiro. I cinque processi sono:

1. **Processo di fornitura**;
2. **Processo di sviluppo**;
3. **Processo di operazione**;
4. **Processo di manutenzione**;

Nota bene: Per la questione didattica il processo di manutenzione non verrà implementato.

1.1 Processo di fornitura

Il processo di fornitura, definito dalla norma *ISO/IEC 12207*$_G$, comprende le attività svolte dal *Fornitore*$_G$ e inizia con la presentazione di una proposta al Proponente o la stipula di un contratto. Nel nostro caso, il processo riguarda la Proponente Zucchetti S.p.A. e comincia al completamento della fase di accettazione della candidatura. Il processo prevede la pianificazione e l'organizzazione delle risorse, delle procedure e dei piani necessari per la gestione del progetto, fino alla consegna del sistema, prodotto o servizio software. Questo processo è fondamentale per garantire che il software soddisfi i requisiti del cliente, sia di alta qualità e venga realizzato rispettando tempi e costi concordati. L'obiettivo principale è allineare costantemente le aspettative dell'acquirente con i risultati ottenuti durante l'esecuzione del progetto. Dunque, per il nostro gruppo, il processo si articola nelle seguenti attività principali:

- **Selezione del capitolato**;
- **Candidatura**;
- **Pianificazione**;
- **Esecuzione e controllo**;

- **Revisione e valutazione**;
- **Consegna e completamento**.

1.2 Selezione del capitolato

Il $Fornitore_G$ va a esaminare i capitolati d'appalto disponibili e in seguito a consultazioni interne raggiunge una decisione unanime sulla candidatura per uno di questi.

1.3 Candidatura

Il $Fornitore_G$ prepara la candidatura al capitolato d'appalto scelto, redigendo i seguenti documenti necessari:

- **Valutazione dei Capitolati**: documento che contiene la valutazione dei vari capitolati d'appalto disponibili, con le motivazioni della scelta;
- **Stima dei Costi e Assunzione Impegni**: documento che contiene un preventivo sulla distribuzione delle ore, la stima dei costi e l'assunzione degli impegni;
- **Lettera di Candidatura**: presentazione del gruppo per il $Committente_G$.

1.3.1 Pianificazione

Il $Fornitore_G$ definisce obiettivi, risorse e procedure necessari per l'esecuzione del progetto, identificando i requisiti per la gestione, la misurazione della qualità e lo svolgimento delle attività. Questa fase riguarda principalmente la stesura del Piano di Progetto, che pianifica l'utilizzo delle risorse e documenta i risultati attesi.

1.3.2 Esecuzione e controllo

Il $Fornitore_G$ attua le attività pianificate nel Piano di Progetto, rispettando le norme stabilite. Inoltre, viene effettuato un controllo continuo sullo stato di avanzamento e sulla gestione delle risorse, garantendo la rendicontazione rispetto agli obiettivi prefissati.

1.3.3 Revisione e valutazione

Il $Fornitore_G$ definisce criteri e procedure per la revisione ed esegue le operazioni in conformità a tali criteri. L'obiettivo è verificare che il progetto soddisfi i requisiti e rispetti gli standard stabiliti.

1.3.4 Consegna e completamento

Consegna del progetto, verifica finale e accettazione da parte dell'acquirente.

1.3.5 Contatti con la Proponente dell'azienda

La Proponente, Zucchetti S.p.A., fornisce l'indirizzo email del proprio rappresentante per la comunicazione asincrona e per la pianificazione di videochiamate su Google Meet con il $Fornitore_G$. Le comunicazioni tra Proponente e $Fornitore_G$ riguardano vari aspetti del progetto, tra cui la raccolta dei requisiti, la raccolta di $feedback_G$ sui risultati ottenuti e le indicazioni sull'avanzamento del progetto. Per ogni colloquio con l'azienda Proponente, ovvero per ogni videochiamata tramite Google Meet, sarà redatto un Verbale Esterno che riassume i punti chiave discussi durante l'incontro.

1.3.6 Documentazione fornita

Di seguito viene descritta la documentazione che il gruppo rende disponibile alla Proponente Zucchetti S.p.A. e ai committenti, Prof. Tullio Vardanega e Prof. Riccardo Cardin.

1.3.6.1 Piano di progetto

Il Piano di Progetto è un documento redatto dal responsabile del progetto che fornisce una visione dettagliata della gestione e dell'organizzazione del gruppo di lavoro. La sua funzione principale è quella di pianificazione, monitoraggio e controllo delle attività, al fine di garantire il raggiungimento degli obiettivi nei tempi e nei costi stabiliti. Inoltre, il piano include l'analisi e la gestione dei rischi, nonché la pianificazione, il preventivo e il consuntivo di ciascuno $sprint_G$, monitorando costantemente l'avanzamento del progetto e le risorse utilizzate. Il documento è suddiviso nelle seguenti sezioni:

- **Introduzione**: una breve descrizione dello scopo del documento e delle varie sezioni che lo compongono;
- **Analisi dei rischi**: riguarda l'identificazione dei potenziali rischi che potrebbero sorgere durante il corso del progetto, i quali potrebbero causare ritardi od ostacoli nella sua progressione. Vengono inoltre sviluppate strategie di prevenzione per evitare che tali rischi si manifestino, nonché strategie di mitigazione per ridurne l'impatto nel caso in cui si verificassero, al fine di garantire la continuità del progetto;
- **Stima dei costi**: calcolo delle risorse necessarie per il completamento del progetto, che viene aggiornato a ogni $sprint_G$;

- **Milestone principali**: sezione dedicata alla descrizione delle $milestone_G$ fondamentali e delle $baseline_G$ di riferimento del progetto;
- **Primo periodo**: Rappresenta la fase iniziale di avanzamento nel progetto;
- **Sprint**: gli $sprint_G$ rappresentano periodi di lavoro di durata fissa in cui vengono definiti obiettivi specifici e attività da completare. Durante ogni $sprint_G$, il gruppo si impegna a raggiungere tali obiettivi. Inoltre, si stabilisce la data per l'inizio dello $sprint_G$ successivo. La sottosezione si articola nelle seguenti parti:
  1. **Obiettivi**: definisce gli obiettivi specifici dello $sprint_G$ elencando le $issue_G$ pianificate per lo stesso;
  2. **Preventivo**: presenta il preventivo dei costi e delle risorse previste per lo $sprint_G$ indicando anche i tempi di fine preventivati per le varie attività, sia in maniera tabellare che tramite un diagramma a torta.
  3. **Consuntivo**: presenta il consuntivo dei costi e delle risorse usate per lo $sprint_G$ indicando anche i tempi di fine delle varie attività, sia in maniera tabellare che tramite un diagramma a torta.
  4. **Retrospettiva**: riporta la conclusione dello $sprint_G$ e valuta i rischi che sono emersi, come sono stati mitigati e se ciò non è avvenuto come verranno mitigati.
  5. **Aggiornamento risorse rimaste**: aggiorna la situazione delle risorse disponibili per il progetto.
  6. **Aggiornamento piano**: aggiorna il piano di progetto con le modifiche apportate durante lo $sprint_G$. Indicandone cambiamenti e motivazioni per esse.
  7. **Aggiornamento rischi**: riporta i rischi che sono emersi durante lo $sprint_G$ e si suddivide nelle seguenti sezioni:
     - rischi incontrati durante lo sprint;
     - rischi che si è riusciti a mitigare;
     - soluzioni adottate per mitigare i rischi;
     - rischi che non sono stati mitigati;
     - soluzioni proposte per mitigare i rischi non mitigati.

1.3.6.2 Analisi dei requisiti

L'analisi dei requisiti è un documento redatto dall'analista che fornisce una visione chiara delle richieste e delle aspettative dell'azienda Proponente. Va a includere un elenco dettagliato delle funzionalità da sviluppare e implementare, nonché i casi d'uso

che definiscono le interazioni tra il sistema e l'utente. Il documento è suddiviso nelle seguenti sezioni:

1. **Introduzione**: Una breve descrizione dello scopo del documento e dei riferimenti usati per la sua stesura;
2. **Descrizione del prodotto**: Descrive gli obiettivi del prodotto, le sue funzioni principali e le caratteristiche;
3. **Requisiti utente**: Elenco completo dei requisiti utente trovati, suddivisi in obbligatori e opzionali;
4. **Use case**: Indica tutti i casi d'uso individuati dal gruppo durante l'analisi;
5. **Requisiti software**: Elenco completo dei requisiti software del prodotto, suddivisi in funzionali e non funzionali(di qualità e di vincolo).

1.3.6.3 Piano di Qualifica

Il Piano di Qualifica è il documento che tratta della specifica degli obiettivi di qualità di prodotto e processo. Definisce un insieme di indici di valutazione e validazione del progetto che verranno usati durante lo svolgimento dello stesso per garantire qualità. Il documento è suddiviso nelle seguenti sezioni:

1. **Introduzione**: Una breve descrizione dello scopo del documento e delle varie sezioni che lo compongono;
2. **Obiettivi di qualità**: Questa sezione presenta i valori accettabili e gli ambiti per le $metriche_G$ definite dal team. Viene divisa per metriche e ogni sotto sezione è dotata di una tabella che descrive le metriche con: ID della metrica, nome, valore tollerabile e valore ambito;
3. **Cruscotto di valutazione della qualità**: Vengono visualizzati i grafici relativi ai valori delle $metriche_G$ e una loro analisi. Il cruscotto di qualità viene aggiornato in maniera periodica per allineare le valutazioni agli $sprint_G$ più recenti.

1.3.6.4 Glossario

Il Glossario è un elenco dettagliato e organizzato di tutti termini, acronimi e definizioni utilizzati nella documentazione. L'obiettivo principale di questo documento è fornire una comprensione chiara dei concetti e dei termini specifici impiegati nel progetto, garantendo una comunicazione coerente e precisa tra tutti i membri del gruppo e con gli $stakeholder_G$. In questo modo si facilita il lavoro collaborativo e si assicura un allineamento efficace su linguaggio e significati durante l'intero $ciclo\ di\ vita_G$ del progetto.

1.3.6.5 Lettera di presentazione

La Lettera di Presentazione è il documento con cui il gruppo comunica la propria candidatura alle revisioni di avanzamento $Requirements\ and\ Technology\ Baseline_G$ e $Product\ Baseline_G$. Nel primo caso essa include informazioni sui $repository_G$ di documentazione e codice sorgente, il riferimento al $Proof\ of\ Concept_G$ e un aggiornamento sugli impegni con la stima del preventivo "a finire".

1.3.6.6 Specifica Tecnica

La Specifica Tecnica è un documento che tratta la progettazione e l'architettura del software del prodotto. Va a descrivere le scelte progettuali e le tecnologie usate per lo sviluppo di quest'ultimo. Viene suddiviso nelle seguenti sezioni:

- **Introduzione**: Una breve descrizione dello scopo del documento e dei riferimenti usati per la stesura;
- **Tecnologie utilizzate**: Descrive le tecnologie utilizzate per lo sviluppo del prodotto;
- **Architettura**: Descrive l'architettura del prodotto, con particolare attenzione ai pattern architetturali usati;
- **Progettazione ad alto livello**: Descrive ad alto livello le varie componenti del sistema e le sue funzionalità;
- **Progettazione del front-end di dettaglio** descrive in dettaglio i componenti del front-end;
- **Progettazione del back-end di dettaglio** descrive in dettaglio i componenti del back-end. Include anche i diagrammi delle classi e i design pattern utilizzati.

1.3.6.7 Manuale Utente

Il Manuale Utente è un documento che fornisce tutte le informazioni necessarie all'utente finale per utilizzare il prodotto software. Il manuale è redatto in modo chiaro e comprensibile, con l'obiettivo di guidare l'utente passo dopo passo nell'utilizzo del software. Il documento è suddiviso nelle seguenti sezioni:

- **Introduzione**: Una breve descrizione dello scopo del documento e dei riferimenti usati per la stesura;
- **Requisiti minimi per l'uso**: Specifica i requisiti minimi necessari per l'utilizzo del software;

- **Installazione**: Guida l'utente nell'installazione del software;
- **Avvio**: Spiega come avviare il software;
- **Utilizzo**: Spiega all'utente come utilizzare il software, con esempi pratici e illustrazioni;

1.3.7 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- LaTeX;
- Git;
- Zoom;
- Meet;
- Google Fogli;
- Google Documenti.

1.4 Processo di sviluppo

Il processo di sviluppo riguarda l'insieme di attività necessarie per produrre e implementare il software, assicurandosi che risponda ai requisiti definiti e sia di qualità adeguata.

1.4.1 Analisi dei requisiti

L'analisi dei requisiti è una attività cruciale nel $ciclo\ di\ vita_G$ del software, in cui vengono raccolte, analizzate e documentate le esigenze degli $stakeholder_G$ per definire chiaramente cosa il sistema dovrà fare. Questa attività produce il documento di analisi dei requisiti descritto nella sezione Documentazione.

1.4.1.1 Requisiti

I requisiti vengono esaminati a due livelli di astrazione differenti e quindi categorizzati in due macro categorie:

1. **Requisiti utente**: descrivono le esigenze degli $stackeholder_G$ dal punto di vista degli utenti del sistema. Questi requisiti vengono estrapolati dal capitolato, dagli incontri interni e dagli incontri con la proponente;

2. **Requisiti software**: descrivono come le caratteristiche che il sistema dovrà avere per soddisfare i primi. Questi requisiti vengono ottenuti tramite l'analisi dei precedenti, quindi un singolo requisito utente viene soddisfatto da più *requisiti software*$_G$. Ogni *requisito software*$_G$ può essere diviso in uno o più sotto requisiti che lo vanno a specificare.

I requisiti di queste macro categorie possono poi essere categorizzati ulteriormente in:

1. **Requisiti funzionali**

   Rappresentano ciò che un sistema deve fare per soddisfare le esigenze degli utenti o degli *stakeholders*$_G$;

2. **Requisiti qualitativi**

   I requisiti di qualità (requisiti non funzionali) descrivono caratteristiche qualitative del sistema, come prestazioni, sicurezza, usabilità e manutenibilità;

3. **Di vincolo**

   I requisiti di vincolo (requisiti non funzionali) rappresentano limitazioni o condizioni obbligatorie che influenzano lo sviluppo, l'implementazione o l'operatività del sistema.

I *requisiti utente*$_G$ vengono identificati usando lo schema:

`RU[priorita]-[identificativo]`

Dove:

1. **priorità**: `O` per obbligatorio e `F` per facoltativo;
2. **identificativo**: numero univoco crescente.

I *requisiti software*$_G$ vengono identificati usando lo schema:

`R[tipo][priorita]-[identificativo].[sotto requisito]`

Dove:

1. **tipo**: `F` per funzionale, `Q` per qualità e `V` per vincolo;
2. **priorità**: `O` per obbligatorio e `F` per facoltativo;
3. **identificativo**: numero crescente univoco all'interno del tipo del requisito.
4. **sotto requisito**: numero crescente univoco all'interno dell'insieme dei sotto requisiti del requisito principale.

1.4.1.2 Casi d'uso

I casi d'uso forniscono una descrizione dettagliata delle funzionalità del sistema dal punto di vista degli utenti, delineando come il sistema deve rispondere a specifiche azioni o scenari. In particolare, un caso d'uso rappresenta un uso completo del sistema dalla prospettiva dell'utente. I casi d'uso vengono rappresentati tramite una combinazione di descrizione e diagramma UML_G e sono usati per approfondire i $requisiti\ utente_G$.

1.4.1.2.1 Descrizione

Nella Tabella 2 viene mostrato il template per la descrizione dei casi d'uso. Ogni descrizione di caso d'uso deve includere:

1. **Identificativo**: indicato usando lo schema `UC-<ID>`. Dove:
   (a) `<ID>`: identificativo numerico del caso d'uso (numero crescente).
2. **Requisito utente**: requisito utente a cui fa riferimento il diagramma;
3. **Pre-condizioni**: stato iniziale del sistema o vincoli necessari per l'attivazione del caso d'uso;
4. **Post-condizioni**: stato finale del sistema dopo un esecuzione normale (priva di errori) del caso d'uso;
5. **Attori**: entità esterna (umana o meno) al sistema che interagisce con il software per iniziare o per contribuire al compimento dell'esecuzione di un caso d'uso. Gli attori si dividono in:
   (a) **Attori primari**: attori principali che partecipano al caso d'uso, solitamente contengono gli attori che iniziano l'interazione e/o sono i destinatari delle informazioni in output del caso d'uso;
   (b) **Attore secondario**: altri attori coinvolti che non sono protagonisti nell'esecuzione del caso d'uso e non sono sotto il controllo diretto del sistema.
6. **Trigger**: evento eseguito da un attore che attiva il caso d'uso;
7. **Scenario principale**: sequenza di step che vengono portati a termine in un esecuzione normale del caso d'uso;
8. **Scenari alternativi**: deviazioni o eccezioni rispetto allo scenario principale.

<identificativo caso d'uso>: <descrizione caso d'uso>	
Requisito utente	<link requisito utente>
Pre-condizioni	• <pre condizione> • <pre condizione>
Post-condizioni	• <post condizione> • <post condizione>
Attore principale	<attore principale>
Attori secondari	<lista attori secondari>
Trigger	<trigger>
Casi d'uso inclusi	<lista casi d'uso inclusi>
Caso d'uso base	<caso d'uso di base>
Scenario principale	1. <1^a azione> 2. <2^a azione> 3. <<include:id>>
Scenari secondari	1.1 <condizione branch dalla 1^a azione> a. 1^a azione del scenario secondario b. 2^a azione del scenario secondario 2.1 <condizione branch dalla 2^a azione> a. 1^a azione del scenario secondario b. 2^a azione del scenario secondario

Tabella 2: Template casi d'uso.

1.4.1.2.2 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono basati sul linguaggio UML_G (Unified Modeling Language), questi diagrammi illustrano i principali scenari operativi del sistema e aiutano

a identificare e chiarire i $requisiti\ funzionali_G$. Ogni diagramma è composto da:

- **Sistema**: delimita i confini del sistema software, indicando quali funzionalità sono incluse e quali sono esterne a esso. Il sistema viene rappresentato nei diagrammi come mostrato in Figura 1;

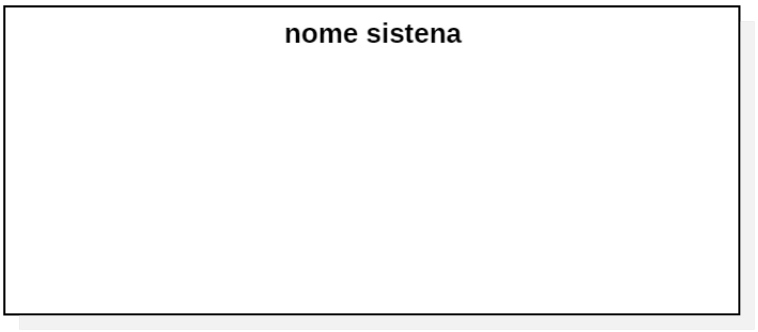

Figura 1: Rappresentazione del sistema in UML_G.

- **Attori**: rappresentano soggetti (persone, altri applicativi o dispositivi) esterni al sistema che vi interagiscono. Un attore viene rappresentato nei diagrammi usando la notazione mostrata in Figura 2 e viene posta al di fuori del sistema;

Figura 2: Rappresentazione degli attori in UML_G.

- **Casi d'uso**: rappresentano le funzionalità offerte dal sistema. Ogni caso d'uso descrive una sequenza specifica di interazioni tra uno o più attori e il sistema. Ogni caso d'uso viene rappresentato nei diagrammi usando la notazione mostrata in Figura 3;

Figura 3: Rappresentazione caso d'uso in UML_G.

- **Relazioni**: notazioni che permettono di rendere più modulari i diagrammi di casi d'uso e di rendere chiara l'interazione degli attori con il sistema.
  - **Associazione**: è la relazione fondamentale che collega un attore a un caso d'uso a cui esso partecipa. Uno stesso attore può partecipare a un numero arbitrario di casi d'uso. Le associazioni vengono rappresentate in UML_G come mostrato in Figura 4;

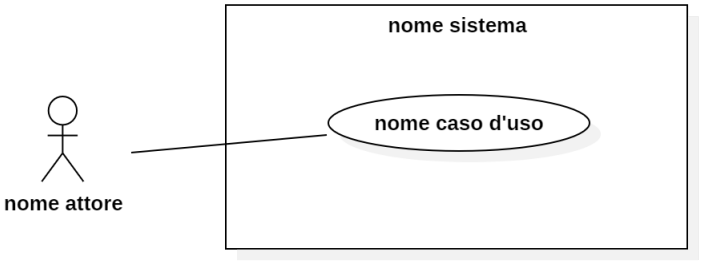

Figura 4: Rappresentazione associazione in UML_G.

  - **Inclusione**: rappresenta una dipendenza in cui il comportamento del caso d'uso incluso è incorporato ogni volta che viene eseguito il caso d'uso base. Il caso d'uso incluso non modifica le post condizioni del caso d'uso che lo include e quindi deve collaborare per raggiungere un obiettivo comune. L'inclusione viene rappresentata in UML_G come mostrato in Figura 5;

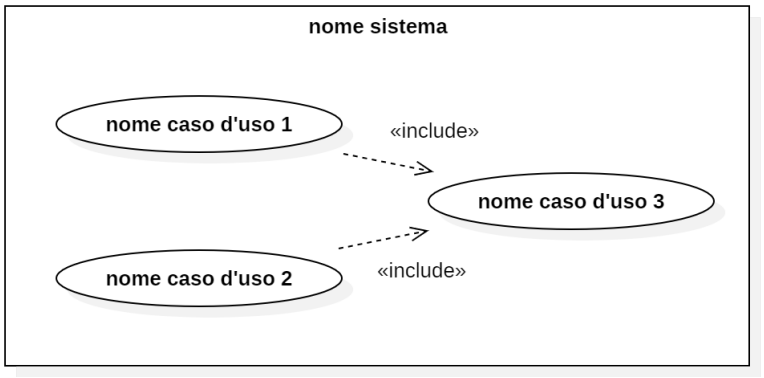

Figura 5: Rappresentazione inclusione in UML_G.

– **Estensione**: mostra una relazione in cui un caso d'uso esteso aumenta le funzionalità del caso d'uso principale e quindi modificando le post condizioni dello stesso. Il caso d'uso esteso viene eseguito solo sotto determinate condizioni, interrompendo l'esecuzione del caso d'uso principale. L'estensione viene rappresentata in UML_G come mostrato in Figura 6;

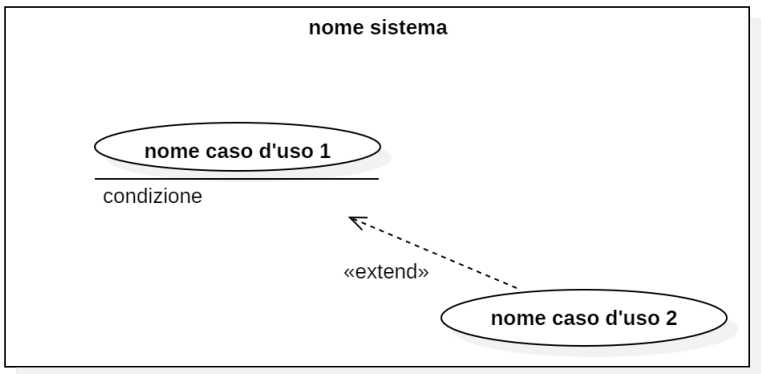

Figura 6: Rappresentazione estensione in UML_G.

– **Generalizzazione**: rappresenta una relazione in cui un caso d'uso figlio può aggiungere funzionalità o modificare il comportamento di un caso d'uso genitore. Tutte le funzionalità definite nel caso d'uso genitore si mantengono nel caso d'uso figlio se queste non vengono ridefinite. La generalizzazione viene rappresentata in UML_G come mostrato in Figura 7;

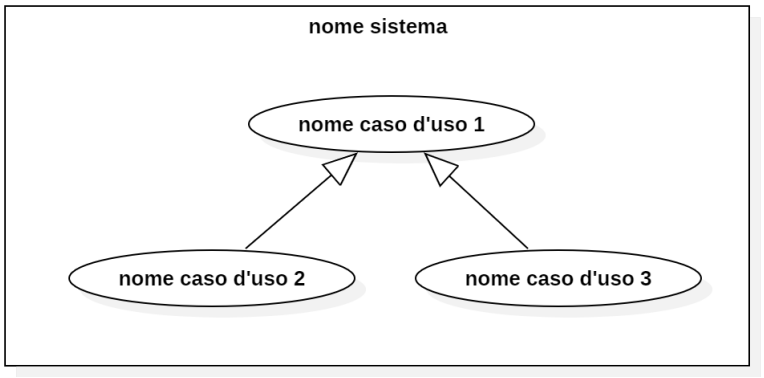

Figura 7: Rappresentazione generalizzazione in UML_G.

- **Sottocasi d'uso**: i sottocasi d'uso rappresentano scenari specifici e dettagliati che si sviluppano all'interno di un caso d'uso principale. La loro funzione è di descrivere in modo approfondito le diverse situazioni o varianti operative che possono verificarsi nel contesto del caso d'uso generale.

1.4.2 Progettazione

La progettazione è l'attività che definisce l'architettura del sistema software, fatta a seguito dell'attività di analisi, che descrive come il sistema soddisferà i requisiti definiti nel documento di *Analisi dei Requisiti*. L'attività di progettazione viene eseguita dal ruolo del progettista, tramite l'uso di pattern architetturali e di design, che permettono di definire la struttura e le relazioni tra le componenti del sistema. Una buona progettazione deve garantire che l'architettura del sistema soddisfi i seguenti aspetti:

- **Affidabilità**: deve garantire che il sistema funzioni nel modo previsto quando viene usato in condizioni previste;
- **Basso accoppiamento**: i moduli devono essere il meno possibile dipendenti tra loro;
- **Flessibilità**: deve avere la capacità di adattarsi a cambiamenti futuri;
- **Sufficienza**: l'architettura deve soddisfare tutti i requisiti definiti nell'analisi;
- **Efficienza**: garantisce un utilizzo efficiente delle risorse;
- **Incapsulamento**: i dettagli di implementazione di un modulo devono essere nascosti agli altri moduli;

- **Semplicità**: evita l'introduzione di elementi superflui;
- **Riusabilità**: deve essere possibile riutilizzare le parti dell'architettura in altri contesti;
- **Modularità**: suddivisione dell'architettura in moduli, in modo che le modifiche a uno di essi non influenzino gli altri;
- **Robustezza**: deve poter gestire situazioni di errore e di eccezione;
- **Coesione**: se esistono elementi che collaborano tra loro, devono essere raggruppati in un unico componente.

1.4.2.1 Fasi della progettazione

La progettazione si suddivide in due fasi:

1. Progettazione logica;
2. Progettazione di dettaglio.

1.4.2.1.1 Progettazione logica

La progettazione logica comprende la prima parte della progettazione del sistema, durante questa viene definita l'architettura del prodotto software. Viene definita la struttura generale del sistema, i moduli principali e le relazioni tra di essi. Per garantire la coerenza e la consistenza dell'architettura, il team andrà a usare stili architetturali che forniscono una soluzione progettuale di alto livello, guidando le scelte strutturali del sistema. Questo approccio assicura il soddisfacimento dei requisiti funzionali e non funzionali, come prestazioni, scalabilità e sicurezza, contribuendo alla manutenibilità e all'evoluzione del sistema nel tempo. La fase di progettazione logica si suddivide in:

- Individuazione delle componenti principali del sistema;
- Definizione delle interfacce e delle relazioni tra le componenti;
- Definizione delle strutture dati necessarie;
- Prima stesura del documento *Manuale Utente*;
- Stesura dei test di integrazione;
- Revisione delle soluzioni e dei pattern architetturali adottati.

1.4.2.1.2 Istruzioni per la progettazione logica del front-end

Per la progettazione logica delle componenti del front-end sono necessari:

- **Descrizione**: descrizione delle funzionalità del componente;
- **Sotto componenti**: lista delle sotto componenti e loro funzione;
- **Tracciamento dei requisiti**: lista dei requisiti che vengono soddisfatti dalla componente.

1.4.2.1.3 Istruzioni per la progettazione logica del back-end

Per la progettazione logica delle componenti del back-end sono necessari:

- **Descrizione**: descrizione delle funzionalità del componente;
- **Informazioni di route**: dettagli relativi alla configurazione e al comportamento delle route del componente;
- **Esecuzione**: descrizione dell'esecuzione della componente e dei suoi possibili output;
- **Tracciamento dei requisiti**: lista dei requisiti che vengono soddisfatti dalla componente.

1.4.2.1.4 Progettazione di dettaglio

La progettazione di dettaglio è la seconda parte della progettazione del sistema, durante questa viene definita la struttura interna delle singole componenti individuate nella progettazione logica. Ciò deve essere fatto prima della codifica del software, in modo da avere una visione chiara di come implementare le funzionalità richieste. L'architettura deve essere tale da consentire a un singolo programmatore di poter codificare una singola unità architetturale. Dunque ogni componente viene suddiviso in parti più piccole, le unità che a loro volta sono composte da moduli. Lo scopo principale della progettazione di dettaglio è quello di organizzare la codifica del software, documentando le scelte di progettazione e collegando i requisiti a ogni singola unità. Il sistema non deve essere però scomposto in unità troppo piccole, in quanto ciò potrebbe portare difficoltà nella coordinazione del lavoro del gruppo. La fase di progettazione di dettaglio si suddivide in:

- Descrizione dettagliata delle singole componenti, delle sue unità e dei moduli;
- Espansione e aggiornamento del documento *Manuale Utente*;
- Espansione e aggiornamento dei test di integrazione;
- Definizione dei test di unità;

1.4.2.1.5 Istruzioni per la progettazione di dettaglio del front-end

Per la progettazione di dettaglio delle componenti del front-end sono necessari:

- **Proprietà**: lista delle proprietà in input alla componente;
- **Eventi**: lista degli eventi in output dalla componente;
- **Aspetti principali**: lista degli aspetti principali della componente.

1.4.2.1.6 Istruzioni per la progettazione di dettaglio del back-end

Per la progettazione di dettaglio delle componenti del back-end sono necessari:

- **Descrizione**: descrizione dettagliata della classe;
- **Diagramma della classe**: diagramma UML della classe;
- **Attributi**: lista (facoltativa) degli attributi della classe;
- **Operazioni**: lista (facoltativa) delle operazioni della classe;
- **Metodi**: lista (facoltativa) dei metodi della classe;
- **Implementazioni**: lista (facoltativa) delle interfacce implementate dalla classe;
- **Dipendenze**: lista (facoltativa) delle dipendenze della classe;
- **Estensioni**: lista (facoltativa) delle estensioni della classe.

1.4.2.2 Specifica Tecnica

Diventa quindi essenziale documentare tramite il documento di *Specifica Tecnica* l'intera fase di progettazione in modo chiaro e dettagliato, poiché sarà il punto di riferimento per la fase di codifica e di testing del software. Il documento di *Specifica Tecnica* va a trattare i seguenti argomenti:

- **Tecnologie utilizzate**: descrizione delle tecnologie, dei framework e delle librerie utilizzate per lo sviluppo del software, motivando le scelte fatte;
- **Architettura generale**: descrizione dell'architettura generale del sistema e del $repository_G$;
- **Progettazione logica**: descrizione delle funzionalità del sistema nel loro complesso, definendone le componenti principali ma non i dettagli;
- **Progettazione di dettaglio**: descrizione dettagliata e tecnica dell'implementazione dei moduli.

1.4.2.3 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- $LaTeX_G$;
- $StarUML_G$.

1.4.3 Testing del codice

Il testing è una fase fondamentale del processo di sviluppo, volta a garantire la qualità e l'affidabilità del software, per questo motivo viene realizzata in parallelo all'attività di codifica. Consiste nell'esecuzione di verifiche sistematiche per individuare errori, incongruenze e difformità rispetto ai requisiti. All'interno del progetto, il testing viene integrato nel flusso di sviluppo e include test automatizzati e manuali per assicurare la conformità del prodotto agli standard definiti. Le categorie di testing eseguite precedentemente all'accettazione del software sono:

- **Test di unità**;
- **Test di integrazione**;
- **Test di sistema**;

1.4.3.1 Test di unità

I test di unità sono una tipologia di test software che verificano il corretto funzionamento delle singole unità o componenti del codice, come funzioni, metodi o classi, in modo isolato. L'obiettivo principale è assicurarsi che ciascuna unità funzioni correttamente secondo le specifiche previste, facilitando l'individuazione precoce di eventuali errori o malfunzionamenti. I test di unità vengono ideati durante la fase di progettazione di dettaglio per poi essere sviluppati in parallelo all'attività di codifica. Per semplificare l'esecuzione dei test di unità vengono utilizzati i seguenti strumenti:

- **Mock object**: oggetti configurabili che imitano il comportamento delle dipendenze reali (come Database o API), consentendo di verificare chiamate ai metodi e risposte attese;
- **Stub**: oggetti che restituiscono risposte predefinite per simulare il comportamento di dipendenze senza eseguire alcuna logica, usati per evitare chiamate reali a servizi esterni;
- **Driver**: moduli che simulano il comportamento di componenti chiamanti non ancora sviluppati.

Esistono tre tipi di tecniche di *unit testing*:

- **Black box testing**: questa tecnica di test viene utilizzata per coprire i test di unità relativi all'input, all'interfaccia utente e alle parti di output, senza considerare la struttura interna del codice;
- **White box testing**: questa tecnica verifica il comportamento funzionale del sistema fornendo input e controllando l'output, includendo l'analisi della struttura interna, del design e del codice dei moduli;
- **Grey box testing**: questa tecnica viene utilizzata per eseguire i casi di test, i metodi e le funzioni di test pertinenti, analizzando le prestazioni del codice nei moduli.

I test di unità vengono eseguiti in maniera automatizzata per garantire che ogni singola componente del software funzioni correttamente in modo isolato. Questo viene fatto attraverso l'utilizzo di strumenti di automazione e framework specifici per i linguaggi di programmazione adottati durante lo sviluppo.

1.4.3.2 Test di integrazione

I test di integrazione, o sono una fase del processo di testing in cui si verificano le interazioni tra diverse unità o moduli del software, per assicurarsi che funzionino correttamente quando combinati. A differenza dei test di unità, che si concentrano su singole componenti isolando il loro comportamento, i test di integrazione verificano che i moduli collaborino come previsto, scambiandosi dati e interagendo senza problemi. Questi test vengono definiti durante la progettazione ad alto livello e hanno lo scopo di rilevare difetti nell'architettura del software o una scarsa qualità dei test di unità. Esistono due principali approcci all'*integration testing*:

- **Bottom-up**: in questo approccio, si parte testando i moduli di basso livello e si lavora verso l'alto, verificando prima i componenti più piccoli e fondamentali. In questo caso, si utilizzano *driver* per simulare i moduli superiori non ancora implementati. Questo approccio è utile quando si vuole testare la stabilità delle basi del sistema prima di testare l'integrazione delle funzionalità più complesse;
- **Top-down**: in questo approccio, i test iniziano con i moduli di alto livello, quelli più vicini alla parte utente o alla logica di business del sistema. Le parti inferiori del sistema (come le dipendenze esterne o i moduli di basso livello) vengono simulate tramite *stub*. Questo approccio è utile quando si vuole verificare l'integrazione delle funzionalità principali, prima di testare i dettagli più specifici.

1.4.3.3 Test di sistema

I test di sistema sono una fase fondamentale del processo di testing in cui si verifica l'intero sistema software nel suo complesso, come una "black box". Vengono eseguiti al completamento dei test di unità e di integrazione e hanno lo scopo di esaminare il software nel suo insieme, testando tutte le funzionalità e le interazioni tra i componenti per garantire che l'intero sistema funzioni come previsto. Questi test includono la verifica dei requisiti funzionali e non funzionali del sistema definiti nel documento di *Analisi dei Requisiti*, come la correttezza delle operazioni, le prestazioni e l'usabilità. I test di sistema possono essere eseguiti in ambienti che simulano il contesto di produzione, assicurando che il sistema possa gestire il carico, le condizioni reali e che tutte le parti siano ben integrate.

1.4.3.4 Notazione dei test

I test vengono indicati all'interno del documento *Piano di Qualifica* e sono identificati così:

```
T.<Tipo>.<Numero>
```

Dove:

- **T**: indica la parola "test";
- **Tipo**: indica la tipologia del test, che può essere:
  - **U**: test di unità;
  - **I**: test di integrazione;
  - **S**: test di sistema;
  - **A**: test di accettazione.
- **Numero**: numero progressivo che identifica univocamente un test per ogni tipologia.

1.4.3.5 Stato dei test

Ogni test indicato all'interno del *Piano di Qualifica* è corredato dallo stato aggiornato alla sua ultima esecuzione, lo stato può essere:

- **S**: test eseguito con successo;
- **F**: test fallito.

1.4.3.6 Linee guida per lo sviluppo dei test

- Testare una sola unità alla volta: ogni test dovrebbe concentrarsi su una singola funzionalità o unità di codice;
- Separare i test dal codice di produzione;
- Ogni test deve essere indipendente dagli altri, se un test fallisce non deve influenzare gli altri test;
- I test, a meno di modifica, devono produrre gli stessi risultati a ogni loro esecuzione;
- Testare limiti, casi estremi e casi di errore.

1.4.4 Codifica

L'attività di codifica ha l'obiettivo di stabilire un insieme di linee guida e standard per la scrittura del codice, in modo da garantire elevati standard qualitativi, la manutenibilità del software e la coerenza tra le differenti componenti del progetto. La codifica rappresenta il momento in cui i requisiti funzionali e tecnici vengono tradotti in implementazioni concrete, chiare e affidabili, ponendo le basi per un sistema robusto e scalabile.

1.4.4.1 Stile comune di codifica

Al fine di assicurare l'uniformità e la leggibilità del codice, si rende indispensabile l'adozione di convenzioni condivise che vadano a disciplinare l'indentazione, la formattazione e l'utilizzo dei commenti. In particolare, si raccomanda:

- L'utilizzo di standard di indentazione e formattazione coerenti, che favoriscano la leggibilità e la comprensione del codice da parte di tutto il gruppo;
- L'impiego di commenti esplicativi e di documentazione inline, finalizzati a descrivere il funzionamento delle porzioni di codice e a facilitare la futura manutenzione;
- L'adozione di convenzioni di nominazione rigorose per variabili, funzioni e classi, in modo da ridurre il rischio di ambiguità e garantire una corretta interpretazione del codice.

1.4.4.2 Struttura del backend

Il componente backend del progetto è sviluppato prevalentemente in Python, avvalendosi del framework Flask per la gestione della logica di business e per l'esposizione di $API\ RESTful_G$. La struttura organizzativa del backend è concepita in modo modulare e prevede:

- Un'organizzazione modulare del codice, che prevede la suddivisione in sottocartelle in base alle funzionalità implementate;
- Le classi dedicate a intercettare e gestire le chiamate HTTP vengono collocate nella cartella `backend/view`, al fine di separare le responsabilità legate all'interfaccia utente e al routing;
- Le classi che si occupano delle operazioni sui dati e dell'interazione con il database sono contenute in `backend/model`, garantendo una chiara separazione dei livelli di accesso ai dati;
- Le classi responsabili della comunicazione e del coordinamento tra i modelli e le view devono essere adibite con metodi di conversione dei tipi, col fine di ridurre la dipendenza tra oggetti secondo l'architettura esagonale.

1.4.4.2.1 Codifica Python

Nel contesto della codifica in Python, si applicano le best practices indicate dal PEP8, garantendo la massima chiarezza e coerenza del codice. In particolare, si raccomanda:

- L'utilizzo di una nomenclatura chiara e coerente per variabili, funzioni e classi, al fine di facilitare la comprensione e la manutenzione del codice;
- Una struttura modulare del codice, che agevoli il riutilizzo delle componenti e la futura estensione delle funzionalità;
- Il posizionamento degli statement `import` all'inizio di ogni file, per garantire una visibilità immediata delle dipendenze e facilitare la gestione del codice.

1.4.4.3 Struttura del frontend

Il modulo frontend è realizzato mediante l'utilizzo del framework Angular, integrando le tecnologie HTML, CSS e TypeScript per la creazione di interfacce utente interattive e reattive. La struttura del frontend prevede:

- L'organizzazione in componenti modulari, che consente una gestione isolata e autonoma delle varie funzionalità dell'interfaccia;

- Una netta separazione tra la logica applicativa, i template HTML e gli stili CSS, in modo da garantire la chiarezza del codice e facilitare il lavoro del gruppo;
- La predisposizione di cartelle dedicate per la gestione degli asset, dei componenti, dei servizi e dei moduli, in conformità con le best practices del framework;
- Il posizionamento dei componenti all'interno della struttura `src/app/...`, in linea con la convenzione standard di Angular.

1.4.4.3.1 Codifica delle componenti `.ts`

Le componenti sviluppate nei file con estensione `.ts` (TypeScript) devono attenersi a linee guida specifiche che ne garantiscano il corretto funzionamento e la facile manutenzione:

- L'utilizzo di classi per definire la logica dei componenti, che permette di sfruttare le potenzialità della programmazione orientata agli oggetti;
- Una chiara separazione tra la logica di business e il markup HTML, assicurando che il codice sia modulare e facilmente testabile;
- L'adozione dei pattern standard di Angular, che facilitano la gestione dello stato dell'applicazione e la comunicazione tra i vari componenti.

1.4.4.3.2 Codifica TypeScript

Per quanto concerne il codice scritto in TypeScript, si raccomanda di:

- Utilizzare la tipizzazione statica per ridurre al minimo la possibilità di errori in fase di compilazione e per garantire una maggiore robustezza del codice;
- Seguire le best practices della programmazione orientata agli oggetti o di quella funzionale, a seconda del contesto applicativo, in modo da scrivere codice chiaro, efficiente e facilmente estendibile;
- Adottare attività di linting (fase di analisi dello stile del codice) e testing, che permettano di individuare tempestivamente eventuali anomalie e di mantenere elevati standard qualitativi.

1.4.4.3.3 Codifica CSS

La scrittura del codice CSS è regolata da linee guida volte a garantire una gestione chiara e modulare degli stili:

- Si raccomanda l'adozione di metodologie di nominazione (es. in base al tipo) per facilitare l'organizzazione e la manutenzione dei fogli di stile;
- È consigliabile la separazione dei file CSS in base alle componenti o alle sezioni della pagina, distinguendo tra stili specifici e quelli relativi al layout generale;
- Gli stili devono essere ottimizzati per garantire la compatibilità coi browser indicati nel documento di *Analisi dei Requisiti* e un design reattivo, in modo da assicurare un'esperienza utente coerente su diversi dispositivi.

1.4.4.3.4 Codifica HTML

Il codice HTML deve essere redatto nel rispetto degli standard internazionali e delle buone pratiche di sviluppo:

- Si raccomanda l'utilizzo di tag semanticamente corretti, al fine di migliorare l'accessibilità e l'ottimizzazione per i motori di ricerca;
- La struttura del documento HTML deve essere ordinata e ben organizzata, facilitando la manutenzione e l'integrazione con altri framework e librerie;
- Adottare le specifiche W3C, garantendo così la validità del markup e la compatibilità con i browser moderni.

1.4.4.4 Codifica SQL

Per quanto concerne la gestione del database, il codice SQL deve essere sviluppato seguendo rigorose linee guida per garantire la sicurezza, l'efficienza e l'integrità dei dati:

- Si devono utilizzare query ottimizzate e parametrizzate per garantire elevate prestazioni;
- È opportuno organizzare le query in moduli o procedure immagazzinate, facilitando così la manutenzione e la gestione delle operazioni sul database;
- Devono essere implementati controlli e validazioni che assicurino l'integrità e la coerenza dei dati in tutte le operazioni di lettura e scrittura;
- Per tabelle potenzialmente voluminose, si raccomanda l'adozione di tecniche di denormalizzazione laddove appropriato, per ottimizzare le prestazioni delle query.

1.4.4.5 Strumenti

Gli strumenti elencati qui in seguito sono approfonditi nella sezione Strumenti:

- Visual Studio Code;
- GitHub.

2 Processi di supporto

2.1 Documentazione

Il processo di documentazione ha lo scopo di registrare le informazioni prodotte da un processo primario garantendo la produzione di documenti coerenti e di qualità. Il processo consiste nelle seguenti attività:

- Implementazione del processo;
- Progettazione e sviluppo;
- Rilascio.

2.1.1 Implementazione del processo

Questa attività stabilisce la tipologia di informazioni contenute in ciascun documento.

- Titolo;
- Scopo;
- Descrizione;
- Responsabilità per redazione, contribuzione, verifica e approvazione;
- Pianificazione per versioni provvisorie e finali.

2.1.2 Progettazione e sviluppo

Questa attività prevede la progettazione e la redazione di ciascun documento, garantendo il rispetto degli standard definiti per formato e contenuto, che verranno successivamente verificati dal responsabile del controllo.

2.1.3 Rilascio

Questa attività inizia con l'approvazione finale del documento da parte del responsabile in carica e, nel caso di verbali destinati all'uso esterno, anche da parte del Proponente. Successivamente, il documento viene pubblicato nell'apposito repository della Documentazione.

2.1.4 Categorie di documenti

Di seguito vengono elencate le due macro categorie di documenti.

2.1.4.1 Documenti interni

Servono al team di lavoro. Sono:

- Verbali interni;
- Norme di progetto;
- Glossario.

2.1.4.2 Documenti esterni

Servono al team di lavoro e alla proponente. Sono:

- Piano di progetto;
- Verbali esterni;
- Analisi dei requisiti;
- Piano di qualifica;
- Specifica Tecnica;
- Manuale Utente.

2.1.5 Ciclo di vita

Il $ciclo\ di\ vita_G$ di un documento è composto dai seguenti eventi:

- Viene individuato il documento o la sua parte da produrre seguendo Implementazione del processo;
- Il redattore elabora una versione del documento conforme alle specifiche e ai metodi definiti in queste norme;

- Il documento viene sottoposto a verifica. Se il verificatore riscontra la presenza di errori che non riguardano il contenuto li risolve. Altrimenti indica al redattore le modifiche contenutistiche da eseguire e il documento ritorna alla fase precedente;
- Dopo un esito positivo della verifica, se il documento è completo e deve essere rilasciato, viene sottoposto all'approvazione finale del responsabile di progetto che come nella fase precedente può richiedere la correzione del contenuto ai redattori.

2.1.6 Standard di formato

I seguenti standard di formato sono validi per tutte le tipologie di documenti. Questi standard devono sottostare agli standard specifici per ogni tipologia di documento indicata nel Piano di documentazione. Gli standard più specifici possono quindi sovrascrivere o specializzare i seguenti standard.

2.1.6.1 Standard di scrittura

- Le tabelle e le immagini devono preferibilmente comparire nella posizione in cui sono specificate all'interno del codice sorgente;
- Le tabelle e le immagini devono essere identificate da una $label_G$ che deve essere usata ogni qual volta sia necessario farvi riferimento;
- Le tabelle e le immagini devono essere accompagnate da una $caption_G$ che ne riassume il contenuto;
- Si devono utilizzare frasi non più lunghe di tre righe;
- I termini che appartengono al glossario devono essere indicati con una G a pedice e in corsivo;
- Si devono usare dei link per fare riferimento a elementi del documento.

2.1.6.2 Standard di forma

2.1.6.2.1 Intestazione

Ogni pagina deve contenere nell'intestazione le seguenti informazioni:

```
Nome documento - <versione>
```

Dove la versione rispetta le regole indicate alla sezione Versione dei documenti.

2.1.6.2.2 Prima pagina

La prima pagina deve contenere le seguenti informazioni:

- Nome documento;
- Nome gruppo;
- Data;
- Versione;
- Logo del gruppo.

2.1.6.2.3 Seconda pagina

La seconda pagina deve contenere il registro delle modifiche per ogni file sottoposto a controllo di configurazione. Il contenuto e la gestione del registro è indicato alla sezione Registro delle modifiche.

2.1.6.2.4 Terza pagina

La terza pagina deve contenere l'indice.

2.1.7 Piano di documentazione

Di seguito viene riportato il piano di documentazione in cui vengono descritte le tipologie di documenti che verranno prodotti durante il $ciclo\ di\ vita_G$ del prodotto e le loro caratteristiche.

2.1.7.1 Verbali interni

2.1.7.1.1 Scopo

I verbali interni sono documenti che hanno lo scopo di registrare il prodotto di una riunione interna al team di lavoro. Permettono quindi di avere una conoscenza condivisa delle decisioni prese e delle cose da fare.

2.1.7.1.2 Autore

L'autore dei verbali interni deve essere il Responsabile.

2.1.7.1.3 Input

Gli input per la stesura di questi documenti derivano direttamente dalla riunione interna all'inizio di ogni $sprint_G$ ovvero ogni lunedì.

2.1.7.1.4 Struttura

I verbali interni sono composti dalle seguenti sezioni e sottosezioni:

1. **Registro presenze**: contiene le seguenti informazioni.
   (a) Data;
   (b) Ora inizio;
   (c) Ora fine;
   (d) Piattaforma usata;
   (e) Tabella che attesta la presenza dei membri del team (intestazioni: Componente e Presenza).
2. **Verbale**: contiene le seguenti informazioni.
   (a) **Ordine del giorno**: decisioni da prendere o azioni da intraprendere;
   (b) **Decisioni prese**: riassunto delle decisioni prese durante la riunione indicando anche una giustificazione.
3. **To do/In progress**: contiene una tabella le cui righe elencano: numero di $issue_G$, breve descrizione, incaricato e data di scadenza per i compiti definiti nella riunione.

2.1.7.1.5 Standard di scrittura specifici

- La data del documento riguarda il giorno in cui è avvenuta la riunione interna.

2.1.7.2 Verbali esterni

2.1.7.2.1 Scopo

I verbali esterni sono documenti che hanno lo scopo di registrare la conoscenza del team sulle necessità della proponente a seguito di una riunione esterna. Permettono quindi la conferma di una conoscenza comune tra team e proponente sulle necessità che il prodotto colma. Funzionano quindi da garanzia al team sul fatto che la sua direzione sia corretta.

2.1.7.2.2 Autore

L'autore dei verbali esterni deve essere il Responsabile.

2.1.7.2.3 Input

Gli input per la stesura di questi documenti derivano direttamente dalla riunione esterna.

2.1.7.2.4 Struttura

I verbali esterni sono composti dalle seguenti sezioni e sottosezioni:

1. **Registro presenze**: contiene le seguenti informazioni.
   (a) Data;
   (b) Ora inizio;
   (c) Ora fine;
   (d) Piattaforma;
   (e) Tabella che attesta la presenza dei membri del team (intestazioni: Componente e Presenza);
   (f) Tabella che attesta i rappresentati della proponente che hanno partecipato alla riunione (intestazioni: Componente e Presenza).
2. **Ordine del giorno**: contiene l'elenco dei dubbi/decisioni da prendere o azioni da intraprendere;
3. **Domande**: contiene la lista delle domande esposte dal team;
4. **Conclusioni**: contiene la lista delle conclusioni che il team ha tratto dalle risposte date dai rappresentati della proponente;
5. **To do/In progress(opzionale)**: contiene una tabella le cui righe elencano: numero di $issue_G$, breve descrizione, incaricato e data di scadenza per gli eventuali compiti definiti nella riunione.
6. **Firma proponente**

2.1.7.2.5 Standard di scrittura specifici

- La data del documento riguarda il giorno in cui è avvenuta la riunione esterna.

2.1.7.3 Analisi dei requisiti

2.1.7.3.1 Scopo

Questo documento ha lo scopo di racchiudere i $requisiti\ utente_G$ e i $requisiti\ software_G$ sul prodotto oggetto del $capitolato_G$.

2.1.7.3.2 Autore

Gli autori di questo documento sono gli Analisti.

2.1.7.3.3 Input

Gli input per la stesura del documento di analisi dei requisiti derivano dall'attività di analisi dei requisiti.

2.1.7.3.4 Struttura

La struttura del documento di analisi dei requisiti è composta dalle seguenti sezioni:

1. **Descrizione prodotto**: descrizione del sistema a un alto livello di astrazione. Questa sezione è composta dalle seguenti sottosezioni:
   (a) **Obiettivi prodotto**: descrive il bisogno che ha portato alla stesura del $capitolato_G$ e come il prodotto le soddisfa;
   (b) **Funzioni prodotto**: descrive le funzioni principali del prodotto;
   (c) **Caratteristiche utente**: descrive gli utenti che useranno il sistema e le loro caratteristiche;
2. **Requisiti utente**: elenca i requisiti utente;
3. **Use case**: mostra gli use case definiti nell'analisi che porta ai $requisiti\ software_G$;
4. **Requisiti funzionali di qualità e di vincolo**: elenca i $requisiti\ funzionali_G$ e non.

2.1.7.4 Norme di progetto

2.1.7.4.1 Scopo

Lo scopo del presente documento è indicato nell'introduzione.

2.1.7.4.2 Autore

L'autore di questo documento è l'Amministratore.

2.1.7.4.3 Input

Gli input per la stesura del documento delle norme di progetto derivano dalle decisioni prese dal team durante gli incontri interni.

2.1.7.4.4 Struttura

Deve esistere una sezione per ogni categoria di processo e una sottosezione per ogni processo indicato nello standard *IEEE 12207:1996*$_G$ usato come guida dal team.

2.1.7.5 Piano di progetto

Il documento piano di progetto contiene le informazioni che permettono la gestione di progetto da parte del Responsabile.

2.1.7.5.1 Autore

L'autore di questo documento è il Responsabile.

2.1.7.5.2 Input

Gli input per la stesura del documento delle norme di progetto derivano dal processo di gestione.

2.1.7.5.3 Struttura

La struttura del documento Piano di Progetto è composta dalle seguenti sezioni:

1. **Analisi dei rischi**: contiene l'output dell'attività di analisi dei rischi;
2. **Stima dei costi**: contiene la stima delle ore che il gruppo ritiene di consumare e i costi che derivano dalle stesse;
3. **Milestone principali**: definisce le *milestone*$_G$ principali, che devono essere sottoposte alla validazione del proponente e/o del *committente*$_G$. Per ogni *milestone*$_G$ vengono indicate:
   - Data di consegna;
   - *Baseline*$_G$;
   - Risorse preventivate.
4. **Primo periodo**: indica l'insieme di risorse utilizzate nel primo periodo di progetto durante il quale la pianificazione era ancora confusionaria. Viene quindi mostrato lo stato delle risorse col termine del primo periodo nelle sottosezioni:
   (a) **Consuntivo**: contiene il consuntivo orario per il primo periodo;
   (b) **Aggiornamento rimaste**: contiene la disponibilità oraria a seguito del primo periodo.

5. **Sprint**: per ogni $sprint_G$ contiene una sottosezione chiamata `sprint n` che a sua volta contiene le seguenti sottosezioni.

   (a) **Obbiettivi**: descrizione degli obbiettivi dello $sprint_G$ tramite una lista puntata contenente la descrizione di essi e le $issue_G$ assegnate a ogni obbiettivo;

   (b) **Preventivo**: contiene il preventivo orario e il preventivo economico per lo $sprint_G$;

   (c) **Consuntivo**: contiene il consuntivo orario e il consuntivo economico per lo sprint;

   (d) **Retrospettiva**: contiene la valutazione dello $sprint_G$ appena concluso, l'elenco e la discussione dei problemi riscontrati e i possibili miglioramenti;

   (e) **Aggiornamento risorse rimaste**: contiene l'aggiornamento delle risorse restanti in seguito allo $sprint_G$;

   (f) **Aggiornamento piano**: contiene le informazioni su eventuali modifiche al piano di progetto. In particolare indica le parti cambiate e le motivazioni dei cambiamenti;

   (g) **Aggiornamento rischi**

2.1.7.6 Glossario

2.1.7.6.1 Scopo

Questo documento ha lo scopo di fornire una definizione condivisa degli acronimi e dei termini tecnici usati nei documenti.

2.1.7.6.2 Autore

L'autore di questo documento è l'Amministratore.

2.1.7.6.3 Input

Gli input per la stesura del documento del Glossario derivano direttamente dalla fase di stesura dei documenti.

2.1.7.6.4 Struttura

Il documento è organizzato come un dizionario in cui i termini vengono indicati raggruppati per iniziale e ordinati in ordine lessicografico.

2.1.7.7 Specifica Tecnica

2.1.7.7.1 Scopo

Questo documento ha lo scopo di descrivere le tecnologie utilizzate nel progetto e i risultati ottenuti dall'attività di progettazione.

2.1.7.7.2 Autore

L'autore di questo documento è il Progettista.

2.1.7.7.3 Input

Gli input per la stesura del documento della Specifica Tecnica derivano soprattutto dall'attività di progettazione.

2.1.7.7.4 Struttura

La struttura del documento Specifica Tecnica è composta dalle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento e i riferimenti.
2. **Tecnologie utilizzate**: descrive le tecnologie utilizzate per l'implementazione del sistema e perché sono state scelte;
3. **Architettura**: descrive l'architettura che verrà utilizzata per il sistema;
4. **Progettazione ad alto livello**: descrive la divisione del sistema in componenti (o moduli) specificando le loro interfacce;
5. **Progettazione di dettaglio**: divide i componenti in classi utilizzando lo strumento dei diagrammi delle classi e ne fornisce una specifica. Descrive la progettazione della base dati utilizzando lo strumento dei diagrammi Entity-Relationship;
6. **Design pattern usati**: elenca i design pattern utilizzati nella progettazione di dettaglio.

2.1.7.8 Manuale Utente

2.1.7.8.1 Scopo

Questo documento ha lo scopo di spiegare agli utenti come installare, avviare e utilizzare il sistema prodotto.

2.1.7.8.2 Autore

L'autore di questo documento è il Progettista.

2.1.7.8.3 Input

Gli input per la stesura del documento Manuale Utente derivano dall'analisi del sistema prodotto.

2.1.7.8.4 Struttura

Il manuale utente è diviso nelle seguenti sezioni:

- **Introduzione**: breve descrizione dello scopo del documento e dei riferimenti usati per la stesura;
- **Requisiti minimi per l'uso**: specifica i requisiti minimi necessari per l'utilizzo del software;
- **Installazione**: guida l'utente nell'installazione del software;
- **Avvio**: spiega come avviare il software;
- **Utilizzo**: spiega all'utente come utilizzare il software, con esempi pratici e illustrazioni;

2.1.8 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti

- Git_G;
- $GitHub_G$;
- $Visual\ Studio\ Code_G$;
- $LaTex_G$;
- $Google\ Docs_G$.

2.2 Gestione della configurazione

Il processo di gestione della configurazione ha il compito di identificare e definire le pratiche e gli strumenti necessari per tenere traccia: delle modifiche, dello stato e dei rilasci dei componenti del sistema.

2.2.1 Repository

I prodotti del progetto vengono nelle seguenti repository:

1. **SorgentiDocumentazione**: https://github.com/ALT-F4-eng/SorgentiDocumentazione;
2. **Documentazione**: https://github.com/ALT-F4-eng/Documentazione;
3. **PoC**: https://github.com/ALT-F4-eng/PoC.

Di seguito vengono approfondito lo scopo e la struttura di tali repository.

2.2.1.1 SorgentiDocumentazione

Questo $repository_G$ contiene i sorgenti della documentazione scritti usando il linguaggio $LaTeX_G$. In particolare contiene le seguenti cartelle e file:

1. **Candidatura**, **RTB** e **PB**: cartelle che contengono le i documenti sorgenti definiti durante le relative fasi del progetto;
2. **Workflows**: file che definiscono le automazioni associate al $repository_G$ e vengono denominati usando la sintassi $Pascal\ case_G$;
3. **Immagini**: cartella che contiene il logo del gruppo;
4. **Packages**: cartella che contiene la definizione del pacchetto $Latex_G$ personalizzato utilizzato dal gruppo;
5. **README.md**: file che contiene la descrizione del $repository_G$ e del gruppo;
6. **.gitignore**: file speciale che contiene una lista di file che il sistema di versionamento deve ignorare.

2.2.1.2 Documentazione

Questo $repository_G$ contiene i file PDF prodotti dalla compilazione dei sorgenti della documentazione e i file necessari per presentarli tramite il sito web del gruppo. Il $repository_G$ contiene le seguenti cartelle e file:

1. **Assets**: cartella che contiene tutte le immagini e le icone usate nel sito web;
2. **Candidatura**, **RTB** e **PB**: cartelle che contengono i PDF dei file sorgente presenti nella relative cartelle del repository SorgentiDocumentazione;
3. **Js**: cartella che contiene tutti i file JavaScript usati dal sito web;
4. **Style**: cartella che contiene tutti i file CSS usati dal sito web;

5. **README.md**: file che contiene la descrizione del $repository_G$ e del gruppo;
6. **index.html**: file html che corrisponde alla home page del sito web.

2.2.1.3 PoC

Questo $repository_G$ contiene i file sorgenti del $Proof\ of\ Concept_G$. Il $repository_G$ contiene le seguenti cartelle e file:

1. **back-end**: cartella che contiene tutti file sorgente che compongono la parte back-end del PoC_G;
2. **front-end**: cartella che contiene tutti i file sorgente che compongono la parte front-end del PoC_G.

2.2.2 Schema di versionamento

Il gruppo ha deciso di utilizzare il seguente schema di versionamento per indicare le versioni dei componenti del sistema:

```
vX.Y
```

Dove:

1. `X`: intero positivo che indica la versione major del componente. La versione major indica a quante volte il componente ha raggiunto uno stato ritenuto come "stabile" dal gruppo;
2. `Y`: intero positivo che indica la versione minor del documento. La versione minor indica il numero di modifiche effettuate all'elemento a partire dall'ultima versione major.

2.2.3 Strategia di branching

Come $strategia\ di\ branching_G$ il gruppo ha scelto l'utilizzo di $Gitflow_G$ dato che permette di massimizzare il lavoro parallelo riducendo la complessità di risoluzione dei conflitti. Di seguito vengono descritti i rami utilizzati da $Gitflow_G$ e le convenzioni usate dal gruppo per la loro denominazione.

1. **main**: contiene la $codeline_G$ principale ovvero le versioni dei file che compongono l'ultima $baseline_G$ raggiunta. Questo ramo è un ramo "stabile" nel senso che persiste nel $repository_G$ lungo tutta la sua vita;
2. **develop**: contiene la $codeline_G$ secondaria ovvero i cambiamenti verificati che si sommeranno formando la prossima $baseline_G$. Anche questo ramo è un ramo "stabile" e viene creato a partire dal ramo main;

3. **feature**: vengono usati dai membri del gruppo per implementare le modifiche. Sono dei rami "provvisori" nel senso che esistono finché il loro contenuto non viene verificato. I rami feature vengono creati a partire dal ramo develop e confluiscono in esso quando il loro contenuto passa con successo la verifica. Il gruppo ha deciso di utilizzare la seguente convenzione per la denominazione di questi rami:

   ```
   feature/#<id issue>
   ```

   Così facendo si ottiene un collegamento immediato tra ramo di feature e richiesta di modifica;

4. **hotfix**: usati per eseguire modifiche rapide e di dimensioni ridotte. Queste modifiche non influenzano quindi le versioni dei componenti del sistema. Questi rami sono "provvisori" nel senso che vengono eliminati dopo che le loro modifiche sono state allineate ai rami main e develop;

5. **release**: usati dal Responsabile del gruppo per approvare il contenuto del ramo develop consolidandolo in una $baseline_G$ nella $codeline_G$ principale. Sono dei rami "provvisori" nel senso che esistono finché il loro contenuto non viene approvato. I rami release vengono creati a partire dal ramo develop e confluiscono nei rami main e develop. Il gruppo ha deciso di utilizzare la seguente convenzione per la denominazione di questi rami:

   ```
   release/<nome milestone>
   ```

   Così facendo si ottiene un collegamento immediato tra ramo di release e la $milestone_G$ che raggiunge.

I comandi base necessari per applicare questa strategia di branching sono:

```
// eseguire commit con messaggio
git commit -m "<messaggio>"

// fusione ramo corrente con ramo indicato
git merge <ramo>

// gestione staging area
git add <lista file>
git rm <lista file>

// importa modifiche da repository remoto
git push origin <ramo>

// pubblica modifiche sul repository remoto
git pull origin <ramo>
```

2.2.4 Gestione delle richieste di modifica

Il gruppo per gestire le richieste di modifica ha deciso di utilizzare le funzionalità offerte da $GitHub_G$.

2.2.4.1 Issue

$GitHub_G$ fornisce un ITS_G ovvero un sistema che permette di registrare e gestire le richieste di modifica. Le richieste di modifica sono rappresentate tramite il concetto di issue. Un $issue_G$ è composta dai seguenti elementi:

1. **Titolo**: nome della richiesta di modifica;
2. **Id**: identifica univocamente la richiesta di modifica;
3. **Stato**: stato attuale della richiesta di modifica tra gli stati che essa può attraversare durante il suo $ciclo\ di\ vita_G$;
4. **Etichetta**: tipologia di richiesta di modifica;
5. **Assegnatario**: membro del gruppo di lavoro che ha la responsabilità di implementare la richiesta di modifica;
6. **Verificatore**: membro del gruppo di lavoro responsabile per la verifica dell'implementazione della richiesta di modifica;
7. **Milestone**: $milestone_G$ a cui la richiesta di modifica appartiene;
8. **Descrizione**: specifica con più precisione la richiesta di modifica.

2.2.4.1.1 Etichette

Di seguito viene fornita una lista delle etichette utilizzate dal gruppo:

1. **bug**: riguarda la correzione di un documento o di un errore presente nel codice;
2. **enhancement**: riguarda l'implementazione di automazioni della gestione e organizzazione di progetto o il miglioramento del codice;
3. **requirement**: riguardano il documento di analisi dei requisiti;
4. **feature**: riguarda l'implementazione di una nuova funzionalità;
5. **documentation**: riguardano lo sviluppo di altri documenti.

2.2.4.1.2 Stato

Lo stato di una $issue_G$ è una proprietà personalizzata che corrisponde allo stato della richiesta di modifica che la $issue_G$ rappresenta. I valori che questa proprietà può assumere sono:

1. **To do**: il membro del team che ha la responsabilità di implementare la richiesta di modifica deve ancora iniziare a lavorarci;
2. **In progress**: il membro del team che ha la responsabilità di implementare la richiesta di modifica ci sta lavorando;
3. **To review**: il verificatore designato deve ancora iniziare la verifica dell'implementazione;
4. **In review**: il verificatore designato sta verificando l'implementazione;
5. **Re open**: il verificatore designato ha ritenuto l'implementazione non all'altezza delle aspettative. L'implementazione deve quindi essere migliorata dal membro del team responsabile;
6. **Done**: il verificatore ha accettato l'implementazione.

2.2.4.2 Ciclo di vita delle issue

Gli stati che una issue attraversa durante il suo $ciclo\ di\ vita_G$ sono riassunti nel diagramma in Figura 8.

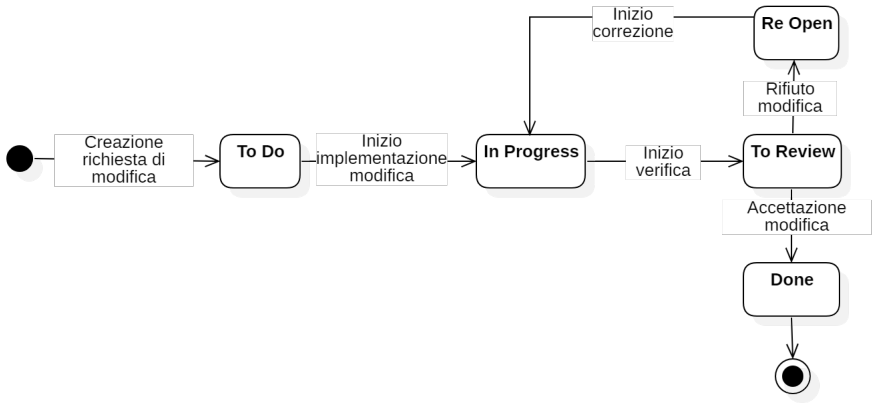

Figura 8: Ciclo di vita di una issue

Per visualizzare l'andamento delle issue definite il gruppo ha deciso di utilizzare una board di progetto che organizza le issue definite in colonne basandosi sul loro stato. Un

membro che ha svolto un compito che fa "avanzare" una issue nel suo ciclo di vita deve quindi interagire con la board. Alcune di queste interazioni sono state automatizzate. Di seguito vengono elenecate le azioni che permettono la modifica dello stato di una issue.

2.2.4.2.1 Richiesta di verifica

Per richiedere la verifica di una modifica è necessario aprire una $pull\ request_G$ dove:

- **Ramo sorgente**: contiene la modifica da verificare.
- **Ramo destinazione**: il ramo develop.

2.2.4.2.2 Presa in carico di una verifica

Per poter prendere in carico una verifica essa deve essere nello stato To Review e quindi risiedere nella relativa colonna della board. Per prendere in carico una verifica il Verificatore deve trascinare la issue dalla colonna To Review alla colonna In Review.

2.2.4.2.3 Accettazione modifiche

Per accettare le modifiche di cui è stata richiesta la verifica il Verificatore deve accettare la $pull\ request_G$ e integrare le modifiche nel ramo delevop. Dopo aver accettato le modifiche il Verificatore deve eliminare il ramo che le conteneva.

2.2.4.2.4 Rifiuto modifiche

Per rifiutare le modifiche di cui è stata richiesta la verifica il Verificatore deve chiudere la relativa $pull\ request_G$. Il verificatore in questo caso ha l'onere di indicare in un commento della $issue_G$ i miglioramenti da apportare alla modifica affinchè venga accettata nella verifica successiva.

2.2.4.3 Registro delle modifiche

Il gruppo ha deciso di dotare alcuni documenti di una tabella chiamata registro delle modifiche. Questo permette di avere una storia delle versioni coerente anche nel caso in cui avvengano errori nella gestione del repository. Di seguito viene indicata la struttura del registro delle modifiche:

1. **Versione**: versione del documento dopo la verifica della modifica;
2. **Data**: data in cui è avvenuta la modifica;
3. **Autore/i**: nome e cognome dei componenti del gruppo che hanno eseguito le modifiche;

4. **Verificatore/i**: nome e cognome dei verificatori (aggiornato solo dopo seguito l'accettazione della modifica);

5. **Descrizione**: lista di link alle sezioni modificate. La descrizione permette ai membri del team di capire velocemente la necessità di allinearsi a nuove informazioni.

Nota bene: Le righe del registro delle modifiche sono ordinate per versione decrescente.

2.2.5 Rilascio

Di seguito viene documentata l'attività di rilascio dei file che devono essere consolidati in una $baseline_G$. Il rilascio deve essere eseguito dal Responsabile del team e prevede le seguenti operazioni:

1. Creazione ramo release seguendo le regole indicate nella sezione Strategia di branching;

2. Eseguire eventuali modifiche correttive minori che non riguardano il contenuto;

3. Registrare le modifiche nel repository locale seguendo le regole indicate nella sezione Strategia di branching;

4. Associare un tag che identifica la baseline raggiunta;

5. Registrare le modifiche nel repository remoto.

2.2.5.1 Compilazione dei documenti sorgente

La compilazione dei documenti avviene in modo automatico a ogni rilascio eseguito nella repository SorgentiDocumentazione. Il meccanismo usato per automatizzare la compilazione è quello delle $GitHub\ action_G$.

2.2.5.2 Distribuzione dei documenti

La distribuzione dei documenti avviene mediante l'utilizzo di un sito web vetrina disponibile al link https://alt-f4-eng.github.io/Documentazione/. Il contenuto del sito viene aggiornato in automatico e quindi rispecchia l'ultima compilazione automatica eseguita.

2.2.6 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- GitHub;
- Git.

2.3 Accertamento della Qualità

2.3.1 Scopo

Il processo di accertamento della qualità ha lo scopo di garantire che i prodotti sviluppati e i processi adottati durante il *ciclo di vita*$_G$ del progetto soddisfino i requisiti stabiliti e rispettino le aspettative previste. Questo processo rappresenta un elemento centrale nella gestione del progetto, in quanto assicura il monitoraggio continuo e la verifica della conformità alle specifiche tecniche, agli obiettivi di progetto e alle normative applicabili. Il processo si basa su principi fondamentali quali tracciabilità, trasparenza e miglioramento continuo. Utilizza inoltre *metriche*$_G$ cioè strumenti oggettivi che permettono di valutare sia l'efficacia dei processi, valutando l'adesione alle *best practices* dell'ingegneria del software, sia la qualità dei prodotti realizzati. Attraverso una pianificazione accurata e un monitoraggio costante, il processo di accertamento della qualità contribuisce a ridurre i rischi, migliorare le performance e a soddisfare le aspettative degli *stakeholder*$_G$.

2.3.2 Garanzia della qualità

Per garantire un efficace accertamento della qualità, il gruppo adotta il *ciclo di Deming*$_G$, noto anche come PDCA (Plan-Do-Check-Act). Questo metodo aiuta a migliorare continuamente processi e prodotti, garantendo il rispetto degli standard richiesti. Il *ciclo di Deming*$_G$ è composto da 4 fasi fondamentali:

- **Plan**: in questa fase si definiscono gli obiettivi di qualità e le strategie per raggiungerli. Vengono quindi stabilite le azioni da intraprendere, le risorse e le tempistiche necessarie per la loro implementazione. Inoltre vengono definite le *metriche*$_G$ da raccogliere durante la fase successiva che permetteranno poi di determinare la qualità del processo e/o del prodotto.
- **Do**: in questa fase, le attività pianificate vengono eseguite seguendo la specifica fornita dal piano. Durante l'esecuzione delle attività vengono raccolti i dati necessari alla fase successiva.

- **Check**: questa fase si verificano i dati raccolti durante la fase "Do" per determinare se i prodotti e i processi sono conformi con gli obiettivi di qualità definiti. In questa fase vengono evidenziati i successi e gli aspetti ancora da ottimizzare. A supporto dell'analisi vengono utilizzati strumenti grafici che permettono di visualizzare l'andamento dei dati e identificare pattern o anomalie.
- **Act**: Durante questa fase i dati ottenuti nelle fasi "Do" e "Check" vengono analizzati per individuare eventuali criticità. Queste possono includere problemi, non conformità, inefficienze, opportunità di miglioramento o qualsiasi altro aspetto che abbia portato a risultati inferiori alle aspettative.

2.3.3 Standard di riferimento per la qualità di prodotto

Per la valutazione della qualità del software con conseguente stesura delle $metriche_G$ viene seguita la normativa $ISO/IEC\ 9126_G$, secondo lo standard le qualità sono suddivise nelle seguenti categorie:

- **qualità esterne**: misurano i comportamenti del software durante la sua esecuzione;
- **qualità interne**: si applicano al software non eseguibile, permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che sia realizzato il software eseguibile;
- **qualità in uso**: rappresentano il punto di vista dell'utente sul software, consentono di stabilire i seguenti obiettivi:
  - *Efficacia*: capacità del software di permettere agli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;
  - *Produttività*: capacità del software di permettere agli utenti di spendere una quantità opportuna di risorse in relazione all'efficacia ottenuta;
  - *Soddisfazione*: capacità del software di soddisfare gli utenti;
  - *Sicurezza*: capacità del software di rimanere entro livelli accettabili di rischi di danni a persone e apparecchiature.

Lo standard normativo stabilisce un modello definendo un set di caratteristiche che consentono di misurare e valutare diversi aspetti della qualità del prodotto software:

- **Funzionalità**: capacità del software di fornire le funzioni adatte a soddisfare le esigenze stabilite;
- **Affidabilità**: capacità del software di mantenere un livello di prestazioni specifico quando usato in date condizioni per un dato periodo di tempo;

- **Efficienza**: capacità del software di fornire adeguate prestazioni relativamente alla quantità di risorse utilizzate;
- **Usabilità**: capacità del software di essere propriamente compreso e utilizzato dall'utente;
- **Manutenibilità**: capacità del software di essere modificato per introdurre migliorie o adattamenti;
- **Portabilità**: capacità del software di essere trasportato da un ambiente di lavoro a un altro.

2.3.4 Notazione Metriche di Qualità

Ogni metrica è identificata in modo univoco seguendo la notazione:

```
M.<Tipo>.<Abbreviazione Nome>
```

Dove:

- **M**: indica "metrica"
- **Tipo**: sarà PC per indicare che la metrica misura la qualità di un processo o PR per indicare che la metrica misura la qualità di un prodotto
- **Abbreviazione Nome**: viene inserito l'acronimo basato sul nome completo della metrica

2.3.5 Didascalia Metriche di Qualità

Le $metriche_G$ sono descritte tramite i seguenti campi:

- **Notazione**: seguendo le indicazioni sopra elencate;
- **Nome**: nome completo della metrica;
- **Descrizione**: descrizione della metrica;
- **Caratteristiche**: presente unicamente nelle $metriche_G$ di prodotto, indica a quale caratteristica, tra quelle indicate nella sezione relativa allo standard di riferimento, si riferisce la metrica;
- **Formula di misurazione**: formula per la misurazione del valore della metrica.

Verranno poi indicati all'interno del documento "Piano di Qualifica" i valori tollerabili e i valori ottimali per ogni metrica e il processo a cui fanno riferimento.

2.3.6 Elenco delle Metriche di Qualità

L'elenco delle $metriche_G$ di qualità da adottare per il progetto è dettagliato secondo la struttura definita e copre diverse aree rilevanti per il processo di accertamento della qualità.

2.3.6.1 Metriche di processo

Planned Value

- **Notazione**: M.PC.PV
- **Descrizione**: rappresenta il costo stimato del lavoro programmato entro un determinato momento del progetto, secondo il piano di progetto originale
- **Formula**:

$$BAC \times (\%LavoroPianificato) \tag{1}$$

  Dove:

  - BAC ($Budget\ at\ Completion_G$): budget previsto per la realizzazione del progetto

Earned Value

- **Notazione**: M.PC.EV
- **Descrizione**: rappresenta il valore del lavoro effettivamente completato alla data corrente
- **Formula**:

$$BAC \times (\%LavoroCompletato) \tag{2}$$

  Dove:

  - $BAC(Budget\ at\ Completion_G)$: budget previsto per la realizzazione del progetto

Actual Cost

- **Notazione**: M.PC.AC
- **Descrizione**: rappresenta il costo sostenuto fino alla data corrente
- **Formula**: costo speso per la realizzazione delle attività svolte fino data corrente

Schedule Variance

- **Notazione**: M.PC.SV
- **Descrizione**: misura la variazione in percentuale tra il valore del lavoro effettivamente completato e il valore del lavoro pianificato. Indica quindi se il progetto è in ritardo o in anticipo rispetto a quanto preventivato
- **Formula**:

$$\frac{EV - PV}{EV} \times 100 \tag{3}$$

Cost Variance

- **Notazione**: M.PC.CV
- **Descrizione**: misura la variazione in percentuale tra il valore del lavoro completato e il costo effettivo sostenuto per tale lavoro. Indica quindi se il progetto sta rispettando il budget pianificato
- **Formula**:

$$\frac{EV - AC}{EV} \times 100 \tag{4}$$

Variazione del piano

- **Notazione**: M.PC.VP
- **Descrizione**: misura la variazione in percentuale tra il numero di task pianificati per un determinato periodo di tempo e il numero di task realmente realizzati. Misura quindi la qualità della pianificazione del lavoro per un periodo di tempo.
- **Formula**:

$$\frac{T_p - T_c}{T_p} \times 100 \tag{5}$$

  Dove:

  - T_p: numero di task pianificati
  - T_c: numero di task completati

Estimated at Completion

- **Notazione**: M.PC.EAC
- **Descrizione**: rappresenta una stima aggiornata del costo totale previsto per la realizzazione del progetto

- **Formula**:

$$AC + (BAC - EV) \tag{6}$$

  Dove:

  - $BAC(Budget\ at\ Completion_G)$: budget previsto per la realizzazione del progetto.

Rischi inattesi

- **Notazione**: M.PC.RI
- **Descrizione**: indica il numero dei rischi che si sono verificati in un determinato periodo e che non erano stati preventivati tramite l'Analisi dei Rischi
- **Formula**: Numero di rischi occorsi e non preventivati

Risk Mitigation Rate

- **Notazione**: M.PC.RMR
- **Descrizione**: indica la percentuale dei rischi occorsi durante lo sviluppo in un determinato periodo che sono stati mitigati correttamente tramite le strategie di mitigazione indicate nella Analisi dei Rischi
- **Formula**:

$$\frac{R_g}{R_t} \times 100 \tag{7}$$

  Dove:

  - R_g: numero di rischi gestiti correttamente
  - R_t: numero totale di rischi verificati nel periodo in analisi

Metriche Soddisfatte

- **Notazione**: M.PC.MS
- **Descrizione**: indica la percentuale di $metriche_G$ soddisfatte, cioè con un valore calcolato che rispetta il valore minimo ammissibile indicato all'interno del Piano di Qualifica
- **Formula**:

$$\frac{M_s}{M_t} \times 100 \tag{8}$$

  Dove:

  - M_s: numero di $metriche_G$ soddisfatte
  - M_t: numero totale delle $metriche_G$ analizzate

2.3.6.2 Metriche di prodotto

Percentuale Requisiti Obbligatori Soddisfatti

- **Notazione**: M.PR.PRM
- **Descrizione**: rappresenta la percentuale di requisiti obbligatori soddisfatti rispetto i requisiti obbligatori totali inseriti all'interno del documento Analisi dei Requisiti
- **Caratteristiche**: Funzionalità
- **Formula**:

$$\frac{RMS}{RMT} \times 100 \tag{9}$$

  Dove:

  - RMS: numero di requisiti obbligatori soddisfatti
  - RMT: numero di requisiti obbligatori totale

Percentuale Requisiti Opzionali Soddisfatti

- **Notazione**: M.PR.PRO
- **Descrizione**: rappresenta la percentuale di requisiti opzionali soddisfatti rispetto al numero totale di requisiti opzionali inseriti all'interno del documento di Analisi dei Requisiti
- **Caratteristiche**: Funzionalità
- **Formula**:

$$\frac{ROS}{ROT} \times 100 \tag{10}$$

  Dove:

  - ROS: numero di requisiti opzionali soddisfatti
  - ROT: numero totale di requisiti opzionali

Correttezza Ortografica

- **Notazione**: M.PR.CO
- **Descrizione**: rappresenta il numero di errori ortografici rilevato all'interno di un documento
- **Caratteristiche**: Usabilità

- **Formula**: N° errori ortografici rilevati

Code Coverage

- **Notazione**: M.PR.CC
- **Descrizione**: misura la percentuale di codice sorgente che è stata eseguita durante i test;
- **Caratteristiche**: Manutenibilità
- **Formula**:

$$\frac{L_{test}}{L_{tot}} \times 100 \tag{11}$$

  Dove:

  - L_{test}: numero di linee di codice testate
  - L_{tot}: numero totale di linee di codice

Branch Coverage

- **Notazione**: M.PR.BC
- **Descrizione**: Misura la percentuale di rami del codice (istruzioni di tipo condizionale come *if, else, switch, ecc.*) che sono stati testati.
- **Caratteristiche**: Manutenibilità
- **Formula**:

$$\frac{R_{test}}{R_{tot}} \times 100 \tag{12}$$

  Dove:

  - R_{test}: numero di rami eseguiti durante i test
  - R_{tot}: numero totale di rami nel codice

Complessità ciclomatica

- **Notazione**: M.PR.COC
- **Descrizione**: Indica la complessità del codice, calcolata sulla base del numero di percorsi indipendenti in una esecuzione con singolo ingresso e singola uscita
- **Caratteristiche**: Manutenibilità

- **Formula**:
$$E - N + 2P \tag{13}$$
Dove:
  - E: numero di archi nel grafico di flusso di controllo
  - N: numero di nodi nel grafico di flusso di controllo
  - P: numero delle componenti connesse da ogni arco

Tempo di risposta

- **Notazione**: M.PR.TR
- **Descrizione**: Il tempo impiegato dal sistema per rispondere a una richiesta.
- **Caratteristiche**: Efficienza
- **Formula**: tempo che intercorre tra la richiesta dell'esecuzione di un test su un dataset e l'ottenimento dei risultati

Browser supportati

- **Notazione**: M.PR.BS
- **Descrizione**: percentuale di browser su cui l'applicazione risulta utilizzabile
- **Caratteristiche**: Portabilità
- **Formula**:
$$\frac{B_s}{B_t} \times 100 \tag{14}$$
Dove:
  - B_s: numero di Browser supportati dall'applicazione
  - B_t: numero totale di Browser

Numero di linee medie di codice per metodo

- **Notazione**: M.PR.LCPM
- **Descrizione**: lunghezza media in termine di linee di codice per metodi o funzioni di una classe
- **Caratteristiche**: Manutenibilità
- **Formula**: calcolato tramite uno script in $Python$

Percentuale di test superati

- **Notazione**: M.PR.PTS
- **Descrizione**: percentuale di test che hanno terminato con successo durante l'esecuzione
- **Caratteristiche**: Funzionalità
- **Formula**:

$$\frac{T_s}{T_t} \times 100 \tag{15}$$

  Dove:

  - T_s: numero di test terminati con successo
  - T_t: numero totale di test

Gestione degli errori

- **Notazione**: M.PR.GE
- **Descrizione**: percentuale di errori gestiti correttamente dall'applicazione
- **Caratteristiche**: Affidabilità
- **Formula**:

$$\frac{E_g}{E_t} \times 100 \tag{16}$$

  Dove:

  - E_g: numero di errori gestiti correttamente
  - E_t: numero totale di errori previsti

2.3.7 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- $LaTeX_G$;
- $Github_G$;
- Git_G;

2.4 Processo di Verifica

Il processo di verifica ha come obiettivo fondamentale l'accertamento che:

- i **prodotti di lavoro** generati durante il $ciclo\ di\ vita_G$ del software rispettino i requisiti specificati
- i **processi seguiti** siano conformi allo standard, alle linee guida e ai piani definiti

2.4.1 Descrizione

Questo processo consiste nel fornire prove oggettive che i risultati di una specifica fase dello sviluppo software rispettino tutti i requisiti previsti. Si basa sull'analisi e revisione del contenuto per valutare la coerenza, la completezza e la correttezza dei risultati. Nel caso di codice, include anche il testing per assicurarsi che i risultati siano conformi alle aspettative definite. Le task fondamentali previste dal processo sono:

- verifica dei processi;
- verifica dei requisiti;
- verifica della progettazione;
- verifica del codice;
- verifica della documentazione.

Per garantire l'accertamento della conformità, ogni volta che si apporta una modifica, è necessario sottoporre l'intero contenuto aggiornato a una verifica. L'incremento della versione del prodotto aggiornato avviene esclusivamente se la modifica viene verificata e la verifica ha esito positivo. Il processo di verifica viene svolto dai membri incaricati come verificatori (che si segneranno all'interno del registro delle modifiche del documento). I quali non possono essere la stessa persona a cui è stata assegnata la realizzazione del prodotto da verificare.

2.4.2 Analisi statica

L'$analisi\ statica_G$ è un approccio alla verifica che non richiede l'esecuzione del codice dell'oggetto di verifica per individuare i difetti del prodotto software e accertarne la sua completezza e coerenza. Si applica non solo al codice, ma anche alla documentazione, verificando la conformità alle regole del prodotto, l'assenza di difetti e la presenza delle proprietà desiderate. Dal team viene utilizzata una tecnica standard per l'$analisi\ statica_G$ dei prodotti: ***Walkthrough*$_G$**.

2.4.2.1 Walkthrough

Il *Walkthrough*$_G$ è uno dei metodi di lettura nell'*analisi statica*$_G$ utilizzato per esaminare e verificare una parte del prodotto, che sia documento o codice, per accertarne la conformità ai requisiti o vincoli stabiliti precedentemente. Si tratta di un approccio collaborativo tra autore e verificatore durante il quale viene esaminato un prodotto o una sua parte, seguendo un percorso prestabilito e cercando di identificare difetti attraverso una lettura critica ad ampio spettro, priva di assunzioni. Nel caso del controllo del codice, il verificatore deve simulare diverse possibili esecuzioni, mentre per i documenti deve analizzarne il contenuto. Le fasi che vengono svolte durante questa tecnica di *analisi statica*$_G$ sono:

- **lettura**: il verificatore effettua una lettura critica dell'oggetto in esame cercando eventuali errori;
- **discussione**: al termine della lettura, nel caso vengano rilevati problemi, il verificatore comunica con gli autori e propone eventuali suggerimenti, con l'obiettivo di correggere i difetti;
- **correzione e repeat**: una volta terminata la discussione e rilevati i difetti, gli autori sono pregati di correggere tali difetti seguendo le indicazioni discusse. Successivamente, si passa di nuovo al passo 2.

2.4.2.2 Verifica della documentazione

La verifica della documentazione, composta solamente da *analisi statica*$_G$ dei documenti realizzati e/o modificati, ha lo scopo di verificare la qualità, completezza, coerenza e conformità dei documenti tecnici relativi al prodotto software da realizzare. Durante l'esecuzione della tecnica di *Walkthrough*$_G$ descritta precedentemente, il verificatore ha lo scopo di assicurarsi che vengano seguiti correttamente gli standard di scrittura e di forma, sia generici che specifici, indicati all'interno del processo di Documentazione. Oltre a ciò, è stata realizzata una *GitHub Action*$_G$ che, all'apertura di una *pull request*$_G$, realizza un'analisi grammaticale dei file modificati e coinvolti nella *pull request*$_G$, fornendo successivamente un *feedback*$_G$ al gruppo di lavoro. La *GitHub Action*$_G$ assicura la correttezza grammaticale dei documenti verificati migliorando il processo di verifica, aumentandone l'efficienza e riducendo il tempo richiesto per le revisioni manuali, permettendo al verificatore di concentrarsi maggiormente su aspetti di forma e di contenuto.

2.4.3 Analisi dinamica

L'*analisi dinamica*$_G$ è un approccio all'analisi di sistemi informatici e prodotti software che si concentra sull'osservazione e verifica del loro comportamento durante l'esecuzio-

ne. Quest'ultima è ampiamente impiegata per individuare errori a runtime, ottimizzare l'efficienza e testare la robustezza contro scenari imprevisti. Grazie alla sua capacità di fornire dati concreti e rilevanti, rappresenta un elemento fondamentale nel processo di sviluppo e manutenzione di applicazioni e sistemi complessi. Essa prevede la definizione di una suite di test, generalmente automatizzati e riproducibili, che vengono eseguiti a runtime per valutare il comportamento del sistema in risposta a specifici input. Questi test verificano la correttezza delle funzionalità, l'efficienza delle prestazioni e l'assenza di errori o anomalie operative. I principali tipi di test che vengono utilizzati per l'$analisi\ dinamica_G$ sono:

- **test di unità**: Mirano a verificare il corretto funzionamento di singole unità o componenti del software (ad esempio, funzioni o metodi). Sono solitamente automatizzati e si concentrano su un ambito ristretto per identificare errori locali;
- **test di integrazione**: Valutano come le diverse unità o moduli del software interagiscono tra loro. L'obiettivo è assicurarsi che le componenti integrate funzionino correttamente come un sistema coerente;
- **test di sistema**: Analizzano il comportamento dell'intero sistema per verificare che soddisfi i requisiti specificati. Considerano il software come un unico blocco, includendo interazioni con l'ambiente e altre applicazioni.

All'interno del processo di Sviluppo, nello specifico nella sezione di testing di codice, vi è una descrizione più approfondita delle varie tipologie di test adottate durante lo sviluppo del prodotto software, ogni test utilizzato verrà poi codificato e indicato all'interno del documento "Piano di Qualifica".

2.4.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti:

- $Visual\ Studio\ Code_G$.
- $GitHub_G$.
- Git_G.
- $LaTex_G$.
- Discord
- Telegram

2.5 Validazione

Il processo di validazione ha lo scopo di accertare che il prodotto finale soddisfi i bisogni dell'utente e l'utilizzo di esso. La validazione viene effettuata come controllo finale, e consiste nel verificare che il prodotto sia conforme alle attese e ai requisiti specificati nel documento *Analisi dei Requisiti*.

2.5.1 Implementazione della validazione

Il raggiungimento della soddisfazione dei test (di unità, di integrazione e di sistema) descritti nel documento di *Piano di Qualifica*, è un requisito necessario per poter procedere con la convalida del software. La validazione deve essere fatta in presenza del referente dell'azienda proponente. Ciò include i seguenti passaggi:

- **Revisione dei requisiti**: il gruppo controlla che tutti i requisiti siano stati soddisfatti;
- **Collaudo**: il gruppo esegue i test di accettazione per verificare che il prodotto implementi correttamente i requisiti e le funzionalità richieste.

Al termine di questi passaggi, il gruppo può procedere con la consegna del prodotto all'azienda proponente che lo convalida.

2.5.2 Test di accettazione

I test di accettazione sono dei test di tipo black-box, cioè si basano sull'esecuzione di specifici casi di test che vanno a riprodurre scenari d'uso realistici, replicando il comportamento degli utenti. Questi vanno, inoltre, a verificare che il prodotto soddisfi i requisiti utente specificati nel documento di *Analisi dei Requisiti*. Poiché questi test mirano a simulare l'esperienza reale dell'utente, è fondamentale che vengano eseguiti su un prodotto completamente funzionante, in modo da garantire un'analisi accurata delle sue prestazioni in condizioni operative effettive.

2.5.3 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- Meet;
- Zoom.

2.6 Risoluzione dei problemi

2.6.1 Scopo

La risoluzione dei problemi è un processo che ha lo scopo di individuare e risolvere le problematiche che si presentano durante lo svolgimento del progetto. Il gruppo deve controllare costantemente lo stato di avanzamento delle attività e, in caso di problemi, deve attivarsi per risolverli nel minor tempo possibile. Ciò deve essere fatto andando a documentarne le cause e le soluzioni adottate, in modo da evitare che si ripresentino in futuro.

2.6.2 Esecuzione

Quando viene riscontrata una problematica all'interno dei prodotti, i membri del gruppo devono segnalarla tempestivamente all'interno del $repository_G$ del progetto con l'etichetta "bug", in modo che possa essere risolta nel minor tempo possibile. Deve essere inoltre notificato il gruppo tramite i canali di comunicazione interni. Il o i membri del gruppo che rilevano il problema devono descriverlo in modo chiaro e dettagliato, in modo che chi dovrà risolverlo possa capire immediatamente di cosa si tratta. Se invece la problematica viene riscontrata durante una riunione interna, questa deve essere documentata all'interno del verbale relativo alla riunione stessa. Quando il problema viene segnalato, questo viene poi assegnato a uno o più membri del gruppo, i quali dovranno risolverlo e documentare la soluzione adottata. Una volta risolto il problema, il responsabile di progetto deve controllare che la soluzione adottata sia corretta e che il problema non si ripresenti.

2.6.3 Hotfix e correzioni minime

Le correzioni minime sono interventi che non richiedono un'analisi approfondita del problema e possono essere risolti in modo rapido ed efficace. Queste modifiche possono essere apportate direttamente dai membri del team che individuano l'errore, senza necessità di assegnazione da parte del responsabile di progetto. Rientrano in questa categoria correzioni di errori di battitura, problemi di formattazione, imprecisioni sintattiche e altre anomalie di lieve entità. L'obiettivo delle correzioni minime è garantire un flusso di lavoro fluido e migliorare la qualità del progetto senza introdurre ritardi significativi.

2.6.4 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- GitHub;

- Zoom;
- Discord;
- Telegram;
- Meet.

2.7 Gestione di progetto

2.7.1 Descrizione

Il processo di gestione racchiude tutte le attività svolte dal responsabile di progetto per il coordinamento del lavoro di tutti i membri del gruppo. Le attività principali che vengono svolte sono:

- Pianificazione;
- Esecuzione e controllo;
- Valutazione e approvazione.

2.7.2 Pianificazione

L'attività di pianificazione consiste nella definizione e organizzazione delle attività da svolgere durante il progetto. Il responsabile di progetto stabilisce le tempistiche, le risorse e le responsabilità. Ogni attività deve avere:

- Stima delle ore necessarie per il suo completamento;
- Data stimata per il completamento;
- Membro incaricato di svolgere l'attività;
- Verificatore incaricato di valutarne il risultato.

Al termine della pianificazione il responsabile e l'amministratore avranno il compito di aprire una $issue_G$ sul progetto di $GitHub_G$ per ogni attività pianificata.

2.7.2.1 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- GitHub;
- Git.

2.7.3 Esecuzione e controllo

L'attività di esecuzione e controllo comprende l'attuazione delle attività pianificate e il monitoraggio del loro andamento. Durante questa fase, il responsabile di progetto coordina il lavoro del team, verificando che le attività vengano svolte secondo i tempi e gli standard definiti. Il controllo continuo permette di identificare eventuali deviazioni dal piano, adottando misure correttive per garantire il raggiungimento degli obiettivi prefissati. Il responsabile avrà il compito di documentare i problemi riscontrati durante lo svolgimento delle attività, le strategie di mitigazione applicate per risolverli e dovrà eventualmente apportare modifiche alla pianificazione se necessario.

2.7.3.1 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- Google Sheets.

2.7.4 Valutazione e approvazione

Terminata ogni attività sarà compito del verificatore esaminare le modifiche apportate a un prodotto e nel caso chiudere la relativa $pull\ request_G$ per integrare nell'ambiente condiviso il lavoro realizzato. Nel caso in cui le modifiche vengano rifiutate, il verificatore è tenuto a indicare nella descrizione della $pull\ request_G$ le modifiche da apportare e le motivazioni del rifiuto. Sarà infine compito del responsabile realizzare l'approvazione finale dei prodotti realizzati, con conseguente $release$, assicurando il raggiungimento degli obiettivi di qualità prefissati.

2.7.4.1 Strumenti

Qui vengono elencati gli strumenti usati, che vengono poi approfonditi nella sezione Strumenti.

- GitHub;
- Git.

2.7.5 Ruoli

Di seguito vengono elencate le definizioni dei ruoli previsti per il progetto didattico e la regola di rotazione.

2.7.5.1 Responsabile

Il responsabile del progetto è incaricato di coordinare le attività del gruppo di lavoro, pianificare e monitorare i progressi, e gestire efficacemente le risorse disponibili. In sintesi, egli si assicura che il progetto venga portato a termine nei tempi stabiliti e in conformità con le risorse assegnate. Ha inoltre il compito di effettuare la redazione del documento *Piano di Progetto* effettuando la stesura del preventivo e del consuntivo per ogni $sprint_G$.

2.7.5.2 Amministratore

L'amministratore ha il compito di effettuare la configurazione e la manutenzione dell'ambiente condiviso. È incaricato della stesura del documento *Norme di Processo*, del calcolo delle metriche di qualità indicate all'interno del documento *Piano di Qualifica* e deve svolgere tutte le attività descritte nel processo di Gestione della configurazione.

2.7.5.3 Analista

L'analista è responsabile dell'analisi delle funzionalità del software, definendo i requisiti e i casi d'uso pertinenti. Ha il compito di effettuare la stesura del documento *Analisi dei Requisiti*.

2.7.5.4 Progettista

Il progettista è responsabile della definizione dell'architettura del software, identificando le componenti e le relazioni tra di esse, sulla base dei requisiti stabiliti dall'analista. La sua attività culmina con la stesura del documento *Specifica Tecnica*

2.7.5.5 Programmatore

Il programmatore si occupa di scrivere il codice sorgente del software e dei test necessari alla sua verifica, implementando l'architettura elaborata dal progettista.

2.7.5.6 Verificatore

Il verificatore di progetto ha il compito di garantire che il software prodotto e la documentazione associata siano conformi alle normative e alle specifiche definite e rispettino gli standard di qualità prefissati.

2.7.5.7 Rotazione dei ruoli

Per aumentare la produttività il gruppo ha deciso di non assegnare staticamente i ruoli a ogni membro durante lo $sprint_G$, a eccezione di:

- Responsabile: sarà incaricato un membro unico che varierà a ogni $sprint_G$;
- Verificatore: incaricato di effettuare la verifica di tutti i prodotti realizzati o modificati con conseguente accettazione o rifiuto della $pull\ request_G$ associata.

Ogni richiesta di modifica pianificata durante lo $sprint_G$ specificherà il ruolo per il quale è competenza svolgerla. Di conseguenza, il membro a cui verrà assegnata la relativa $issue_G$ assumerà il ruolo a essa associato per il tempo necessario al suo compimento. Così facendo è possibile massimizzare l'efficienza del gruppo, infatti ogni membro ha la possibilità di ottimizzare il tempo a disposizione realizzando richieste di modifica di competenza di ruoli diffetenti.

2.7.6 Gestione dei rischi

La gestione dei rischi è curata dal responsabile di progetto e viene effettuata al termine di ogni $sprint_G$, con la registrazione dei rischi riscontrati all'interno del consuntivo di periodo. Questo processo contribuisce inoltre alla redazione dell'analisi dei rischi, riportata nel documento $Piano\ di\ Progetto$, garantendo un monitoraggio costante e un miglioramento continuo delle strategie di mitigazione.

2.7.6.1 Notazione dei rischi

Ogni rischio verrà indicato con la sigla

```
R[Tipo][Numero]
```

Dove:

- **R**: indica la parola "rischio";
- **Tipo**: indica la tipologia del rischio che può essere:
  - **O**: rischio organizzativo;
  - **T**: rischio tecnologico.
- **Numero**: numero progressivo che identifica univocamente un rischio per ogni tipologia.

2.7.6.2 Didascalia dei rischi

I rischi individuati verranno descritti nel seguente modo:

- **descrizione**: descrizione del rischio;
- **pericolosità**: può essere bassa, media o alta e indica l'impatto che avrebbe sul progetto il suo verificarsi;

- **probabilità**: può essere bassa, media o alta e indica la probabilità del rischio di verificarsi;
- **prevenzione**: indica la metodologia utilizzata per identificare il rischio;
- **mitigazione**: azioni da intraprendere quando si verifica il rischio.

2.7.7 Canali di comunicazione

I canali di comunicazione si dividono in canali interni usati per le riunioni del gruppo e canali esterni usati per le riunioni tra il gruppo e il proponente.

2.7.7.1 Canali di comunicazione interni

La comunicazione tra membri del gruppo avviene utilizzando le seguenti tecnologie:

1. **Telegram (comunicazione asincrona)**: applicazione di messaggistica che permette comunicazioni rapide di interesse generale e di contenuto breve;
2. **Discord (comunicazione sincrona)**: utilizzata per effettuare chiamate di gruppo sia formali che informali. Permette la creazione di diversi canali di testo per argomenti specifici e di diversi canali vocali per permettere anche una comunicazione interna tra membri del gruppo.

2.7.7.2 Canali di comunicazione esterni

La comunicazione tra i membri del gruppo e l'azienda proponente avviene tramite i seguenti canali di comunicazione:

1. **Gmail (comunicazione asincrona)**: verrà utilizzato principalmente per la condivisione di file/documenti e per organizzare chiamate sincrone tra gruppo e azienda proponente;
2. **Google Meet (comunicazione sincrona)**: utilizzata per effettuare videochiamate per il contatto diretto con l'azienda proponente, durante le quali verranno discussi eventuali dubbi dei membri del gruppo o verrà presentato il lavoro svolto per ricevere un $feedback_G$.

2.8 Processo di miglioramento

2.8.1 Scopo

Questo processo definisce le procedure operative per l'analisi, la valutazione e il miglioramento continuo del processo di sviluppo software, al fine di garantire elevati

standard di qualità e l'efficienza dei processi. Lo scopo è quello di definire, controllare e migliorare i processi di *ciclo di vita*$_G$ del software.

2.8.2 Attività

Le attività operative da eseguire sono:

1. Definizione dei processi;
2. Valutazione dei processi;
3. Miglioramento dei processi.

2.8.3 Definizione dei processi

Vengono analizzati i processi di *ciclo di vita*$_G$ del software definiti dallo standard *ISO\IEC 12207:1996*$_G$ e vengono individuati i processi necessari alla realizzazione del progetto. Successivamente viene stabilito il *Way of Warking*, vengono quindi documentate le norme necessarie per implementare correttamente i processi individuati all'interno del documento *Norme di Progetto*.

2.8.4 Valutazione dei processi

Il responsabile di progetto ha il compito di valutare i processi implementati durante l'ultimo *sprint*$_G$ di lavoro. L'obiettivo è quello di individuare eventuali criticità sull'implementazione dei processi e assicurarne la loro efficacia individuandone le aree migliorabili. Per fare questo, il responsabile deve inoltre calcolare le *metriche*$_G$ definite all'interno del *Piano di Qualifica* che permettono, per determinati processi, di registrarne l'andamento, ottenendo una valutazione più accurata.

2.8.5 Miglioramento dei processi

Una volta identificati e valutati i processi implementati durante lo sviluppo e individuati gli aspetti migliorabili nell'implementazione dei processi, è necessario aggiornare le *Norme di Progetto* per avvicinarsi incrementalmente allo stato dell'arte.

2.8.6 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti:

- *LaTeX*$_G$;
- *GitHub*$_G$;

- Discord;
- Telegram.

3 Strumenti

Nella seguente sezione sono elencati gli strumenti utilizzati dal team per lo svolgimento delle attività di progetto.

3.1 Discord

Link - https://discord.com/
Il team utilizza la piattaforma Discord per le riunioni interne e per condividere materiale utile allo svolgimento del progetto

3.2 Telegram

Link - https://web.telegram.org/
Il team utilizza la piattaforma Telegram per la messaggistica istantanea e viene utilizzata principalmente per condividere informazioni o in caso di dubbi

3.3 Latex

Link - https://www.latex-project.org/
Il gruppo utilizza Latex per redigere tutti i documenti ufficiali presenti nel $repository_G$

3.4 Git

Link - https://git-scm.com/
Il gruppo ha scelto di utilizzare Git come version control system

3.5 GitHub

Link - https://github.com/
Il gruppo ha deciso di utilizzare GitHub come piattaforma di hosting per lo sviluppo software collaborativo. GitHub offre controllo di versione, $repository_G$ Git, gestione del progetto e automazioni.

3.6 Google Docs

Link - https://docs.google.com/
Il gruppo ha deciso di utilizzare Google Docs per scrivere documenti non ufficiali online in modo collaborativo in tempo reale.

3.7 Gmail

Link - https://mail.google.com/
Il gruppo ha scelto di utilizzare Gmail come servizio di posta elettronica per la comunicazione ufficiale con i professori e l'azienda proponente.

3.8 Google Sheets

Link - https://docs.google.com/spreadsheets/
Il gruppo ha deciso di utilizzare Google Sheets sia per scrivere in fogli di calcolo non ufficiali online in modo collaborativo in tempo reale sia per organizzare dati in modo strutturato e automatico attraverso le github actions.

3.9 Google Slides

Link - https://docs.google.com/presentation/
Il gruppo ha deciso di utilizzare Google Slides per la creazione dei diari di bordo e per le presentazioni relative alle revisioni di avanzamento.

3.10 StarUML

Link - https://staruml.io/
Il gruppo ha deciso di utilizzare StarUML per la creazione di tutti i diagrammi UML presenti nella documentazione.

3.11 Visual Studio Code

Link - https://code.visualstudio.com/
Il gruppo ha deciso di utilizzare Visual Studio Code come editor sia per la documentazione che per la codifica.

3.12 LTeX+

Link - https://ltex-plus.github.io/ltex-plus/
Il gruppo ha deciso di utilizzare LTeX+ per il controllo grammaticale dei documenti.

3.13 Google Meet

Link - https://meet.google.com/
Il gruppo ha deciso di utilizzare Google Meet come piattaforma di teleconferenza per gli incontri tra il team, i professori e il proponente.

3.14 Zoom

Link - https://www.zoom.com/
Il gruppo ha deciso di utilizzare Zoom come piattaforma di teleconferenza per gli incontri tra il team, i professori e il proponente.