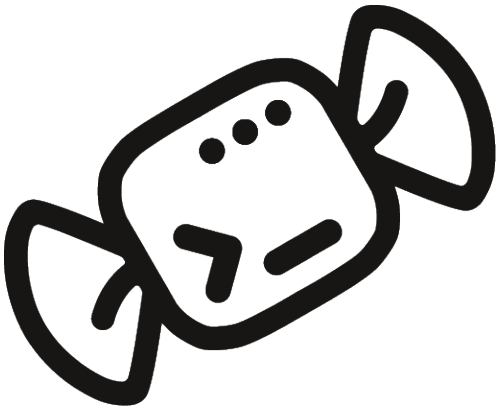

SWEetCode

# Norme di progetto


**Componenti del gruppo**

Bresolin G.
Campese M.
Ciriolo I.
Dugo A.
Feltrin E.
Michelon R.
Orlandi G.



Università di Padova
Ingegneria del Software

## Registro delle versioni

| Versione | Data | Responsabile di stesura | Revisore | Dettaglio e motivazioni |
|---|---|---|---|---|
| v2.27.8(1) | $2024-04-09$ | Feltrin E. | Bresolin G. | Aggiornamento strumenti Codifica, Testing e Integrazione Software. |
| v2.27.7(17) | $2024-04-09$ | Bresolin G. | Michelon R. | Aggiunta sezione Integrazione software. |
| v2.27.7(14) | $2024-04-06$ | Bresolin G. | Orlandi G. | Aggiornamento sezione Progettazione: modifiche documentazione e strumenti. |
| v2.16.5(1) | $2024-03-13$ | Ciriolo I. | Campese M. | Ristrutturazione con specifica di procedure. |
| v2.9.0(1) | $2024-03-09$ | Campese M. | Ciriolo I. | Aggiornamento sezione Testing del codice. |
| v2.2.15(1) | $2024-03-07$ | Campese M. | Ciriolo I. | Aggiornamento normativa progettazione logica e di dettaglio. |
| v2.0.0(38) | $2024-03-02$ | Campese M. | Bresolin G. | Correzione linee guida consuntivo. |
| v2.0.0(10) | $2024-02-29$ | Campese M. | Feltrin E. | Aggiornamento linee guida di preventivo e consuntivo del Piano di progetto. |
| v2.0.0(4) | $2024-02-26$ | Campese M. | Dugo A. | Modifiche da linguaggio narrativo a procedurale. |
| v2.0.0(0) | $2024-02-11$ | - | - | Approvazione versione finale RTB. |
| v1.10.4(14) | $2024-02-09$ | Campese M. | Michelon R. | Aggiornamento e correzione sezione Metriche. |

Continua nella pagina successiva

| Versione | Data | Responsabile di stesura | Revisore | Dettaglio e motivazioni |
|---|---|---|---|---|
| v1.10.4(13) | $2024-02-09$ | Campese M. | Michelon R. | Modifiche correttive e aggiornamento riferimenti. |
| v1.10.1(25) | $2024-01-14$ | Bresolin G. | Michelon R. | Aggiornamento strumenti. |
| v1.10.1(16) | $2024-01-04$ | Michelon R. | Bresolin G. | Introduzione Accertamento della qualità. |
| v1.10.1(15) | $2024-01-04$ | Michelon R. | Bresolin G. | Introduzione Risoluzione dei problemi. |
| v1.10.1(14) | $2024-01-04$ | Michelon R. | Bresolin G. | Introduzione Infrastruttura. |
| v1.10.1(9) | $2024-01-03$ | Bresolin G. | Orlandi G. | Introduzione Miglioramento. |
| v1.10.1(8) | $2024-01-03$ | Bresolin G. | Orlandi G. | Introduzione Revisione. |
| v1.10.0(15) | $2023-12-29$ | Bresolin G. | Orlandi G. | Introduzione Revisione congiunta. |
| v1.10.0(13) | $2023-12-29$ | Bresolin G. | Orlandi G. | Introduzione Validazione. |
| v1.10.0(12) | $2023-12-29$ | Michelon R. | Campese M. | Introduzione Processi organizzativi. |
| v1.10.0(11) | $2023-12-29$ | Michelon R. | Campese M. | Introduzione Verifica. |
| v1.10.0(10) | $2023-12-29$ | Michelon R. | Campese M. | Introduzione Codifica. |
| v1.10.0(9) | $2023-12-29$ | Michelon R. | Campese M. | Introduzione Progettazione. |
| v1.10.0(5) | $2023-12-26$ | Bresolin G. | Michelon R. | Aggiornamento configuration management. |
| v1.10.0(4) | $2023-12-26$ | Ciriolo I. | Bresolin G. | Rimozione sezione Overleaf per cambio strumenti. |
| v1.10.0(3) | $2023-12-21$ | Bresolin G. | Ciriolo I. | Aggiornamento: sviluppo. |

Continua nella pagina successiva

| Versione | Data | Responsabile di stesura | Revisore | Dettaglio e motivazioni |
|---|---|---|---|---|
| v1.9.0(9) | $2023-12-18$ | Ciriolo I. | Bresolin G. | Processi primari: Acquisizione. |
| v1.9.0(8) | $2023-12-18$ | Ciriolo I. | Bresolin G. | Processi primari: Fornitura. |
| v1.9.0(7) | $2023-12-18$ | Bresolin G. | Ciriolo I. | Introduzione: applicazione Standard ISO/IEC 12207:1995. |
| v1.9.0(6) | $2023-12-17$ | Bresolin G. | Ciriolo I. | Introduzione: riferimenti normativi e informativi. |
| v1.9.0(5) | $2023-12-14$ | Ciriolo I. | Bresolin G. | Introduzione: glossario. |
| v1.0.0(15) | $2023-12-08$ | Bresolin G. | Orlandi G. | Aggiornamento ITS: passaggio a Jira di Atlassian. Motivazione: GitHub non offre un diagramma di Gantt in grado di stabilire le dipendenze tra i vari tasks; maggior supporto al framework Scrum attraverso le funzionalità presentate. |
| v1.0.0(11) | $2023-12-02$ | Ciriolo I. | Orlandi G. | Processo primario: Sviluppo: attività Analisi dei requisiti. |
| v1.0.0(10) | $2023-11-27$ | Campese M. | Orlandi G. | Ampliamento sezioni Comunicazioni e Strumenti. |
| v1.0.0(2) | $2023-11-22$ | Bresolin G. | Campese M. | Aggiornamento branch protection rules. |
| v1.0.0(1) | $2023-11-22$ | Michelon R. | Bresolin G. | Aggiunte sezioni automazione release e versionamento documenti. |

Continua nella pagina successiva

| Versione | Data | Responsabile di stesura | Revisore | Dettaglio e motivazioni |
|---|---|---|---|---|
| v0.0.1(23) | $2023-11-20$ | Michelon R. | Campese M. | Aggiunta parametrizzazione template documenti. |
| v0.0.1(22) | $2023-11-13$ | Bresolin G. | Campese M. | Aggiornamento struttura di documento: modifica ordine registro delle versioni dalla più recente e aggiunta voce "motivazioni"; verbali: introduzione #issue nella tabella delle azioni. |
| v0.0.1(21) | $2023-11-13$ | Bresolin G. | Campese M. | Aggiornamento documentazione: struttura di documento: presenza obbligatoria registro delle versioni e rimozione versione; norme tipografiche: aggiornamento versioni. Aggiunta configuration managment: version control. |
| v0.0.1(20) | $2023-10-30$ | Bresolin G. | Michelon R. | Processi di supporto: configuration management: *ITS* e pull request; Processi organizzativi: *GitHub Teams*. |
| v0.0.1(19) | $2023-10-30$ | Campese M. | Michelon R. | Documentazione: strutture di documento: posta elettronica. |
| v0.0.1(18) | $2023-10-30$ | Dugo A. | Bresolin G. | Processi organizzativi: comunicazione. |
| v0.0.1(13) | $2023-10-28$ | Ciriolo I. | Campese M. | Strumenti. |

Continua nella pagina successiva

| **Versione** | **Data** | **Responsabile di stesura** | **Revisore** | **Dettaglio e motivazioni** |
|---|---|---|---|---|
| v0.0.1(8) | $2023-10-27$ | Bresolin G. | Ciriolo I. | Processo di supporto: documentazione e verifica. |
| v0.0.1(5) | $2023-10-26$ | Campese M. | Orlandi G. | Introduzione. |

# Indice

# 1 Introduzione

## 1.1 Obiettivo del documento

L'obiettivo che ci si pone nella realizzazione di questo documento è di normare e documentare un corretto *Way of Working*$_{|g|}$ che verrà seguito lungo l'intero svolgimento del progetto.
Tutti i membri del team sono tenuti a visionare tale documento e a seguire le norme contenute in esso. Tali norme permettono di perseguire e migliorare la coerenza ed omogeneità del lavoro e dei file prodotti. Viene inoltre fornito uno storico delle modifiche e delle versioni del documento per seguire un approccio incrementale nella sua redazione, facilitandone i cambiamenti nel corso del tempo.

## 1.2 Glossario

Per evitare ambiguità ed incomprensioni relative al linguaggio e ai termini utilizzati nella documentazione del progetto viene presentato un Glossario. I termini ambigui o tecnici-specifici presenti nello stesso, vengono identificati nei corrispondenti documenti con un pedice |g| e con una scrittura in corsivo. All'interno dei documenti viene identificata con tale scrittura solo e soltanto la prima occorrenza presente nel testo di un termine definito nel Glossario.

# 2 Riferimenti

## 2.1 Riferimenti normativi

- *ISO/IEC 12207:1997 Information technology — Software life cycle processes*
  `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf`
  (Ultimo accesso: 2024-02-08);
- *Regolamento del progetto didattico*
  `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf`
  (Ultimo accesso: 2024-02-08);
- *SEMAT*
  `https://semat.org/documents/20181/57862/formal-18-10-02.pdf/formal-18-10-02.pdf`
  (Ultimo accesso: 2024-02-08).

## 2.2 Riferimenti informativi

- *Capitolato C1*: *Knowledge Management AI*
  - `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C1.pdf`
    (Ultimo accesso: 2024-02-08);
  - `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C1p.pdf`
    (Ultimo accesso: 2024-02-08).
- Dispense Ingegneria del software 2023/2024:
  - `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T2.pdf`
    (Ultimo accesso: 2024-02-08);

- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T3.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T4.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T5.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T7.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T8.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T9.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T10.pdf` (Ultimo accesso: 2024-02-08);
- `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T11.pdf` (Ultimo accesso: 2024-02-08).

- *(Glossario v3.0.0(0))*.

# 3 Applicazione standard ISO/IEC 12207:1997

In questo documento il team si impegna ad essere conforme allo Standard textitISO/IEC 12207:1997 - Information technology — Software life cycle processes. Questa sezione del documento è dedicata all'applicazione pratica degli standard e dei processi definiti nell'ambito di questo Standard Internazionale.

## 3.1 Organizzazione dello Standard: processi del ciclo di vita del software

In questa sezione viene presentata l'organizzazione dei processi secondo lo *Standard ISO/IEC 12207:1997 - Information technology — Software life cycle processes* del *ciclo di vita del software*$_{lgl}$. In questo documento utilizzato per normare il *way of working* del team, viene presentata una organizzazione gerarchica in cui ogni *processo*$_{lgl}$ è costituito da un insieme di attività, che a loro volta possono presentare delle procedure e un elenco di strumenti utilizzati nel loro svolgimento.
Al termine del documento, come ultima sezione non prevista dallo *Standard ISO/IEC 12207:1997*, il team presenta un elenco completo di tutti gli strumenti menzionati all'interno di questo documento, accompagnandoli con un riferimento e una breve descrizione.

### 3.1.1 Processi primari

Sebbene lo Standard adottato presenti cinque processi primari (*Acquisizione, Fornitura, Sviluppo, Operazione, Manutenzione*), all'interno del contesto del progetto universitario in atto, il team non ritiene i processi *Manutenzione* e *Acquisizione* ( quest'ultimo di competenza del committente) pertinenti; pertanto, si è deciso di escluderli dalla presentazione nel documento.
I processi primari presentati nel presente documento sono:

- **Fornitura**: definisce le attività del fornitore, l'organizzazione che fornisce il prodotto software all'acquirente;
- **Sviluppo**: definisce le attività dello sviluppatore, l'organizzazione che definisce e sviluppa il prodotto software.

### 3.1.2 Processi di supporto

I processi di supporto presentati nel presente documento sono:

- **Documentazione**: definisce le attività per la registrazione delle informazioni prodotte da un processo del ciclo di vita;
- **Configuration *Management*$_{lgl}$**: definisce le attività di gestione della configurazione;
- **Assicurazione della qualità**: definisce le attività per assicurare in modo oggettivo che i prodotti e i processi software siano conformi ai *requisiti*$_{lgl}$ specificati e rispettino i piani stabiliti;
- ***Verifica*$_{lgl}$**: definisce le attività per verificare il prodotto software;
- ***Validazione*$_{lgl}$**: definisce le attività per validare il prodotto software;

- **Revisione congiunta**: definisce le attività per valutare lo stato e i prodotti di un'attività. Questo processo può essere utilizzato da qualsiasi coppia di parti, in cui una parte (la parte che esegue la revisione) valuta un'altra parte (la parte che viene revisionata) in un forum congiunto;
- **Revisione**: definisce le attività per determinare la conformità ai requisiti, piani e contratti;
- **Risoluzione dei problemi**: definisce un processo per analizzare e risolvere i problemi di qualsiasi natura o origine, scoperti durante l'esecuzione di processi.

### 3.1.3 Processi organizzativi

I processi organizzativi presentati nel presento documento sono:

- **Gestione organizzativa**: definisce le attività di management durante i processi del ciclo di vita;
- **Infrastruttura**: definisce le attività base per stabilire una struttura in un processo del ciclo di vita;
- **Miglioramento**: definisce le attività che il team esegue per stabilire, misurare, controllare e migliorare i processi del ciclo di vita;
- **Formazione**: definisce le attività atte a provvedere una adeguata formazione del team.

### 3.1.4 Ruoli

I ruoli definiti all'interno di questo progetto didattico universitario sono:

- **Docente del corso**: *committente*$_{lgl}$;
- **Azienda proponente**: cliente e mentore;
- **Gruppo di lavoro**: fornitore.

# 4 Processi primari

## 4.1 Fornitura

### 4.1.1 Scopo

Il processo di fornitura descrive le attività svolte dal fornitore e coinvolge pianificazione, acquisizione e gestione delle risorse necessarie. Il processo segue la determinazione delle procedure e delle risorse necessarie per gestire e garantire il progetto. L'obiettivo del processo è la garanzia dell'*efficienza*$_{lgl}$ e della conformità ai requisiti del progetto per raggiungere gli obiettivi stabiliti con il proponente.

### 4.1.2 Implementazione di processo

Il processo di fornitura è formato dalle seguenti 7 fasi:

- **Iniziazione**: Il fornitore effettua una revisione dei requisiti nella richiesta di proposta;
- **Preparazione della risposta**: Il fornitore definisce e prepara una proposta in risposta alla richiesta di proposta;
- **Negoziazione**: Il fornitore negozia e stipula un accordo con il proponente per fornire il prodotto finale;
- **Pianificazione**: Il fornitore effettua una revisione dei requisiti di acquisizione e valuta le opzioni per lo sviluppo del prodotto software in base ad un'analisi dei rischi associati a ciascuna opzione per definire la struttura del piano di gestione del progetto e per garantire la qualità del prodotto finale (il fornitore deve sviluppare e documentare tale piano);
- **Esecuzione e controllo**: Il fornitore esegue il piano di gestione del progetto e monitora e controlla il progresso e la qualità del prodotto durante l'intero ciclo del progetto;
- **Revisione e valutazione**: Il fornitore coordina le attività di comunicazione con il proponente e conduce o supporta riunioni informali, revisioni di accettazione, test di accettazione, revisioni congiunte. Il fornitore esegue la verifica e la convalida del processo per dimostrare che i prodotti o servizi software e i processi soddisfano appieno i rispettivi requisiti;
- **Consegna e completamento**: Il fornitore consegna il prodotto finale, e dovrà fornire assistenza al proponente a supporto del prodotto consegnato.

### 4.1.3 Gestione

Per garantire un processo di fornitura efficace ed efficiente viene mantenuta la comunicazione con l'azienda proponente durante tutto il corso del progetto, mediante riunioni settimanali su *Google Meet*$_{lgl}$ pianificate con *Google Calendar* e scambio di messaggi qualora fosse necessario attraverso la piattaforma *Slack*$_{lgl}$. In questo modo al fornitore risulta possibile un'identificazione accurata dei bisogni del proponente con conseguente individuazione accurata dei requisiti e dei vincoli del progetto. Il dialogo continuo con il proponente permette infine una valutazione continua e costante del la-

voro svolto dal fornitore, in modo da permettere correzioni, integrazioni e miglioramenti in modo incrementale e costruttivo.

#### 4.1.4 Piano di Progetto

Questo documento viene redatto dal responsabile con l'aiuto degli amministratori. Fornisce una guida dettagliata per la pianificazione, l'esecuzione e il controllo del progetto e serve come base per il monitoraggio del progresso, la gestione dei rischi e la comunicazione tra proponente e fornitore.
Il suo contenuto comprende:

- Calendario di progetto;
- Stima dei costi di realizzazione;
- Rischi e mitigazione;
- Pianificazione e modello di sviluppo;
- Preventivo e consuntivo;
- Retrospettiva.

##### 4.1.4.1 Linee guida stesura preventivo di periodo

Le sezioni di preventivo documentano i preventivi per ogni periodo fino ai successivi due a partire da quello corrente. Il preventivo si compone delle seguenti sottosezioni:

- **Preventivo delle ore e dei costi**: preventivo delle ore e dei costi associati corredato di:
  - Tabella 'Preventivo ore';
  - Grafico 'Istogramma della ripartizione oraria';
  - Tabella 'Preventivo costi';
  - Grafico 'Areogramma della ripartizione oraria dei ruoli'.
- **Previsione occorrenze eventuali rischi**: in questa sezione si cercano di anticipare le probabili occorrenze dei rischi descritti nel (Piano di progetto, §Rischi e loro mitigazione).

##### 4.1.4.2 Linee guida stesura consuntivo di periodo

Le sezioni di consuntivo documentano i consuntivi di periodo per ogni sprint concluso e si compongono delle seguenti sottosezioni:

- **Consuntivo delle ore e dei costi**: consuntivo ore e costi effettivi consumati corredato di:
  - Tabella 'Consuntivo ore';
  - Tabella 'Consuntivo costi'.
- **Revisione**: sezione contenente:
  - Elenco delle attività effettivamente portate a termine nel periodo;

- **Rischi occorsi e loro mitigazione**: se sono occorsi rischi (previsti o meno), questa sezione contiene riflessioni sull'efficacia delle misure di mitigazione associate;
- **Pianificazione futura**: valutazioni su come questo consuntivo di periodo influenzi la pianificazione futura di task/milestone;
- **Preventivo a finire**: descrizione di come questo consuntivo impatti il preventivo originale.

- **Retrospettiva**: problemi e riflessioni riguardanti il lavoro svolto:
  - Identificazione di cosa è andato bene in modo da poterlo consolidare;
  - Segnalazione di cosa è andato male e come si può migliorare applicando il ciclo *PDCA*.

#### 4.1.5 Piano di Qualifica

Il (Piano di qualifica v3.0.0(0)) definisce le strategie, gli obiettivi e le attività per garantire la qualità del prodotto finale. Vengono definite in tale documento le metriche di valutazione e validazione del progetto e la specifica degli obiettivi di qualità del prodotto finale. Tali parametri vengono stabiliti in accordo ai requisiti e alle aspettative del proponente e talvolta a discrezione del team sulla base delle valutazioni fatte nel corso di studi. Il suo scopo è assicurare che il prodotto soddisfi gli standard di qualità definiti e che eventuali deviazioni vengano gestite in modo appropriato.
Il suo contenuto comprende:

- Qualità di processo;
- Qualità di prodotto;
- Test e specifiche;
- Valutazioni per il miglioramento;
- Resoconto delle attività di verifica.

#### 4.1.6 Strumenti

- *Google Calendar*;
- *Google Sheet*;
- *Slack*.

### 4.2 Sviluppo

#### 4.2.1 Scopo

Il processo di sviluppo si pone lo scopo di descrivere tutte le attività e i compiti che portano un prodotto dalla fase di ideazione a quella di realizzazione.

#### 4.2.2 Sviluppo

Il processo di sviluppo presentate le seguenti attività:

- Analisi dei requisiti;

- Progettazione logica;
- Progettazione di dettaglio;
- *Codifica*$_{lgl}$;
- Testing;
- Integrazione software;
- Installazione software.

### 4.2.3 Analisi dei requisiti

#### 4.2.3.1 Scopo

Lo scopo dell'attività di analisi dei requisiti è comprendere e definire in modo completo e accurato le esigenze e le aspettative dell'azienda proponente e degli utenti relativamente al prodotto software.

#### 4.2.3.2 Implementazione dell'attività

L'attività di analisi dei requisiti, la quale viene documentata all'interno del documento (Analisi dei requisiti v3.0.0(0)), viene svolta seguendo le seguenti fasi:

- Studio accurato del capitolato e delle esigenze del committente;
- Individuazione dei casi d'uso (e relativa produzione dei diagrammi dei casi d'uso) e dei requisiti;
- Discussione con il proponente del materiale prodotto;
- Divisione dei requisiti nelle tre categorie individuate e applicazione degli spunti emersi dalla discussione col proponente.

(L'analisi dei requisiti, essendo una attività incrementale, prevede che alcune fasi siano svolte più volte all'interno del progetto). Il risultato di tale processo è un documento che contiene tutti i requisiti richiesti dal proponente, con relativi casi d'uso dal punto di vista dell'utente finale.

#### 4.2.3.3 Casi d'uso: Notazione

I casi d'uso sono indicati con la notazione seguente: **UC[Codice] - [Titolo]**
in cui:

- **UC** sta per *Use Case*$_{lgl}$;
- [**Codice**] è l'identificativo del caso d'uso. È composto da un unico numero progressivo univoco assegnato in base all'ordine di esplorazione, se il caso d'uso non ha padre, mentre se si tratta di un sottocaso d'uso, segue il formato **[Codice_padre].[Numero_figlio]**; questa struttura è ricorsiva, quindi non pone un limite alla profondità della gerarchia;
- [**Titolo**] è il titolo del caso d'uso.

#### 4.2.3.4 Casi d'uso: Didascalie

Per ogni caso d'uso è presentata una breve didascalia che ne indica:

- *Attore*$_{lgl}$ principale (attore che partecipa e dialoga attivamente con il sistema nello scenario principale per raggiungere uno scopo);
- Attore secondario (attore esterno al sistema che, se presente, aiuta il sistema a raggiungere l'obiettivo dell'attore principale);
- Precondizioni (condizioni che devono essere vere affinché il caso d'uso possa essere eseguito);
- Postcondizioni (condizioni del sistema dopo che il caso d'uso si è concluso);
- Scenario principale (sequenza di passaggi che definisce un'interazione tra attore e sistema);
- Scenario alternativo (scenario che si verifica se si diverge dallo scenario principale);
- Estensioni (lista di casi d'uso che deviano il flusso principale del caso d'uso in esame, introducendo flussi alternativi);
- Inclusioni (lista di casi d'uso che rappresentano una parte dello scenario principale, ma estratti come casi d'uso singoli per aumentare il riuso nel caso in cui appaiono in scenari principali di più casi d'uso);
- Generalizzazioni (lista di casi d'uso che ereditano le caratteristiche del caso d'uso in esame aggiungendo specificità);
- Trigger (evento scatenante del caso d'uso).

#### 4.2.3.5 Requisiti: Notazione

Ogni requisito preso in analisi sarà trattato con la sigla: **R[Tipo].[Importanza].[Codice]** nella quale:

- [**R**] indica la parola "requisito";
- [**Tipo**] può essere:
  - F (Funzionale);
  - Q (Qualità);
  - V (Vincolo);
  - V (Prestazione).
- [**Importanza**] classifica i requisiti in:
  - O (Obbligatorio);
  - D (Desiderabile);
  - OP (Opzionale).
- [**Codice**] identifica i requisiti per ogni tipologia. È composto da un unico numero progressivo univoco assegnato in ordine di importanza se il requisito non ha padre, mentre se si tratta di un sotto-requisito, segue il formato **[Codice_padre].[Numero_figlio]**;

questa struttura è ricorsiva, quindi non pone un limite alla profondità della gerarchia;

#### 4.2.3.6 Requisiti: Suddivisione

- I requisiti funzionali descrivono le funzionalità del sistema, le azioni che il sistema può compiere e le informazioni che il sistema può fornire.
  Seguendo la notazione riportata sopra, questi si possono partizionare in:
  - RF.O - Requisito Funzionale Obbligatorio;
  - RF.D - Requisito Funzionale Desiderabile;
  - RF.OP - Requisito Funzionale Opzionale.
- I requisiti di qualità descrivono come un sistema deve essere, o come il sistema deve esibirsi, per soddisfare le esigenze dell'utente.
  Seguendo la notazione riportata sopra, questi si possono partizionare in:
  - RQ.O - Requisito di Qualità Obbligatorio;
  - RQ.D - Requisito di Qualità Desiderabile;
  - RQ.OP - Requisito di Qualità Opzionale.
- I requisiti di vincolo descrivono i limiti e le restrizioni normative/legislative che un sistema deve rispettare per soddisfare le esigenze dell'utente.
  Seguendo la notazione riportata sopra, questi si possono partizionare in:
  - RV.O - Requisito di Vincolo Obbligatorio;
  - RV.D - Requisito di Vincolo Desiderabile;
  - RV.OP - Requisito di Vincolo Opzionale.
- I requisiti di prestazione descrivono vincoli di tempo e spazio che il prodotto deve rispettare.
  Seguendo la notazione riportata sopra, questi si possono partizionare in:
  - RP.O - Requisito di Prestazione Obbligatorio;
  - RP.D - Requisito di Prestazione Desiderabile;
  - RP.OP - Requisito di Prestazione Opzionale.

#### 4.2.3.7 Diagrammi

I diagrammi dei casi d'uso e di attività riportati all'interno del documento (Analisi dei requisiti v3.0.0(0)) devono seguire la notazione e le specifiche indicate da *UML$_{lgl}$ v2.5*.

#### 4.2.3.8 Applicazione milestone SEMAT

SWEetCode nell'esecuzione dell'attività di analisi dei requisiti ha deciso di adottare la pianificazione in *milestone$_{lgl}$* presentata da *SEMAT*: tale pianificazione viene però piegata alle esigenze e alla compatibilità di questo progetto didattico universitario, motivo per il quale le milestone presenti saranno:

- *RTB-AdR*: *Bounded*: i bisogni macro sono chiari e i meccanismi di gestione dei requisiti sono fissati;

- *RTB-AdR*: *Coherent*: i requisiti sono classificati e quelli obbligatori sono chiari e ben definiti;
- *PB-AdR*: *Acceptable*: i requisiti fissati definiscono un sistema soddisfacente per l'utente;
- *PB-AdR*: *Fulfilled*: il prodotto soddisfa abbastanza requisiti da meritare la piena approvazione dell'azienda proponente.

#### 4.2.3.9 Strumenti

- *Diagrams.net*.

### 4.2.4 Progettazione

#### 4.2.4.1 Scopo

L'attività di progettazione ha lo scopo di fissare un'architettura del prodotto che soddisfi i requisiti determinati dall'attività di analisi, prima di passare alla codifica. La progettazione serve a dominare la complessità del prodotto, decomponendolo in parti componibili e organizzate.

#### 4.2.4.2 Documentazione

L'attività di progettazione produce il documento (Specifica tecnica (v3.0.0(0))). Questo documento serve a descrivere l'architettura del prodotto e a guidare in modo chiaro e privo di ambiguità il lavoro dei programmatori attraverso la progettazione di dettaglio. Il documento tratta le seguenti tematiche:

- **Scelte tecnologiche**: analisi delle tecnologie e delle librerie selezionate e delle motivazioni a supporto di queste scelte;
- **Architettura di sistema**: descrizione del modello architetturale e organizzazione delle componenti al suo interno;
- **Architettura delle componenti**: esposizione dell'organizzazione delle componenti ed enunciazione delle loro sottocomponenti;
- **Progettazione di dettaglio**: specifica dettagliata di ogni modulo.

#### 4.2.4.3 Linee guida progettazione delle componenti

- Ogni componente deve essere descritta come segue:
  1. Descrizione concisa delle sue funzionalità;
  2. Caratteristiche:
     - **Route API**;
     - **Metodo** (tipo);
     - **Lista parametri HTTP**.
  3. Possibili esiti della sua esecuzione corredati di:
     - Codice;
     - Descrizione;

- Risposta (messaggio da esibire nel caso di quel particolare esito).

4. Riferimenti che permettano di fare tracciamento dei requisiti che essa implementa;
5. Enunciazione completa delle sue sottocomponenti e del loro scopo.

#### 4.2.4.4 Linee guida progettazione di dettaglio

Il documento Specifica tecnica (v3.0.0(0)) nella sezione (§Progettazione di dettaglio) analizza in maniera precisa e chiara i moduli, raggruppandoli per appartenenza alle componenti del sistema.
Per ogni componente viene allegato un *Diagramma delle classi*.
Per ogni classe vengono elencate le proprietà che le caratterizzano, tra le quali:

- **Attributi**: elenca gli attributi propri della classe;
- **Implementazione**: campo che specifica se la classe implementa un'interfaccia;
- **Estensione**: campo che specifica se la classe estende un'altra classe;
- **Metodi**: ogni metodo viene identificato dalla firma e da una descrizione in cui viene eventualmente indicato se è astratto;
- **Valori**: campo che caratterizza le enumerazioni;

Le proprietà sono presentate nell'elenco solamente nel caso in cui ci siano elementi associati alla classe in questione.

#### 4.2.4.5 Convenzioni nella progettazione di dettaglio

- Firma del metodo:
  `nomeMetodo(parametro0: tipoParametro0, parametro1: tipoParametro1): TipoDiRitorno`

**4.2.4.6 Pattern** I progettisti del team avranno l'obbligo di considerare le seguenti proprietà durante l'attività di progettazione:

- **Affidabilità**: svolge bene il suo lavoro;
- **Basso accoppiamento**: limita le dipendenze tra parti diverse;
- **Coesione**: parti che agiscono sulle stesse unità di dati sono raccolte nello stesso componente;
- **Efficienza**: è efficiente nello spazio, nel tempo e nei consumi;
- **Flessibilità**: permette modifiche mantenendo limitati i costi per apportarle;
- **Incapsulazione**: nasconde l'interno delle componenti architetturali all'esterno;
- **Modularità**: suddivisa in parti chiare e ben distinte, esponendo interfacce e nascondendo l'implementazione;
- **Riusabilità**: permette il riuso delle sue parti in applicazioni esterne;
- **Robustezza**: resiliente a più tipi e quantità di input;
- **Semplicità**: ogni parte contiene lo stretto necessario;
- **Sufficienza**: in grado di soddisfare tutti i requisiti;

#### 4.2.4.7 Scelte tecnologiche e PoC

- Responsabile e amministratori effettuano le scelte tecnologiche;
- Vengono analizzate le caratteristiche, i pro e i contro delle opzioni tecnologiche disponibili, trovando una soluzione e considerando i requisiti prodotti dall'analisi dei requisiti;
- Progettisti e *programmatori*$_{lgl}$ validano la soluzione tramite realizzazione di un Proof of Concept, il cui scopo è:
  - Dimostrare la fattibilità del progetto;
  - Dimostrare che le scelte tecnologiche fatte sono adeguate e compatibili;
  - Avere feedback concreto e veloce da parte dell'azienda proponente;
  - Eslorare concretamente le tecnologie scelte.
- Progettisti e programmatori si assumono il compito di realizzare più PoC, qualora fosse necessario un confronto concreto tra tecnologie.

#### 4.2.4.8 Fasi di progettazione

Seguendo il *modello a V*, l'attività di progettazione si divide in due fasi: progettazione logica e progettazione di dettaglio.

#### 4.2.4.9 Progettazione logica

La progettazione logica consiste nel trasformare i requisiti software in un'*architettura*$_{lgl}$ che descrive la struttura di alto livello del prodotto software e ne identifica i componenti. In questa fase i progettisti devono assicurarsi che tutti i requisiti siano assegnati ai componenti e che questi siano ulteriormente dettagliati, in modo da facilitare la successiva progettazione di dettaglio.

In particolare, i progettisti devono:

- Progettare le componenti del software;
- Definire e progettare le strutture dati necessarie per soddisfare i requisiti;
- Documentare una versione preliminare del (Manuale utente);
- Pianificare e definire, collaborando con i verificatori, i test di sistema del software;
- Valutare e revisionare l'architettura definita nei passi precedenti, coinvolgendo tutti i membri del team e altre parti interessate. In particolare, il team collaborerà con dei membri dell'azienda proponente in modo da ottenere consigli e correzioni da fonti autorevoli e con esperienza.

#### 4.2.4.10 Progettazione di dettaglio

La progettazione di dettaglio consiste nel suddividere il sistema affinché ogni singola parte abbia una complessità tale da poter essere codificata da un unico individuo, in modo rapido, fattibile e verificabile.

La progettazione di dettaglio agisce sulle unità architetturali, cioè unità funzionali ben definite, la cui codifica può essere assegnata ad un unico programmatore. Ogni unità

architetturale è formata da uno o più moduli, e più unità architetturali formano un componente, che è l'oggetto dell'architettura logica.

Durante la progettazione di dettaglio, i progettisti hanno il compito di:

- Definire e progettare in modo dettagliato i componenti software, lavorando a livello di unità e moduli;
- Aggiornare e dettagliare il (Manuale utente);
- Definire la specifica dei test di unità, collaborando con i verificatori;
- Definire la specifica dei test di integrazione, collaborando con i verificatori.

#### 4.2.4.11 Diagrammi

I diagrammi delle classiriportati all'interno del documento (Specifica tecnica (v3.0.0(0))) devono seguire la notazione e le specifiche indicate da *UML v2.5*.

#### 4.2.4.12 Strumenti

- *StarUML*.

### 4.2.5 Codifica

#### 4.2.5.1 Scopo

La codifica, svolta dai programmatori, ha lo scopo di realizzare concretamente il prodotto software seguendo l'architettura definita dai progettisti. Questa sezione ha il compito di normare il codice, in modo da:

- Facilitarne la lettura, aumentarne l'uniformità e la robustezza;
- Facilitarne e velocizzarne la verifica;
- Facilitarne la manutenzione, il *debugging*$_{lgl}$ e l'estensione;
- Migliorarne la qualità.

#### 4.2.5.2 Stile di codifica: Python

##### 4.2.5.2.1 Struttura dei file

La struttura di un sorgente *Python*$_{lgl}$ è definita dalle seguenti parti:

- Import: insieme di istruzioni di import che seguono l'ordine:
  1. *Librerie esterne*$_{lgl}$;
  2. *Librerie interne*$_{lgl}$.
- *Classe*$_{lgl}$: rappresenta un'unità o raccoglie funzioni di utilità.

##### 4.2.5.2.2 Struttura delle classi

La struttura di una classe Python è definita dalle seguenti parti, se necessarie:

1. Lista di classi estese o interfaccia implementata;
2. *Costruttore*$_{lgl}$;

3. *Attributi*$_{lgl}$;
4. *Metodi*$_{lgl}$.

Ogni classe deve essere opportunamente documentata, seguendo il formato *Google docstrings*$_{lgl}$; in particolare:

- Dopo la definizione della classe, devono essere presenti, in una serie di commenti:
  - Descrizione del ruolo della classe;
  - Lista degli argomenti richiesti dal costrutture, seguiti da una breve descrizione;
  - Lista degli attributi della classe, seguiti da una breve descrizione;
  - Lista dei metodi della classe, seguiti da una breve descrizione.
- Dopo la definizione dei metodi, devono essere presenti, in una serie di commenti:
  - Descrizione del metodo;
  - Lista di argomenti del metodo, seguiti da una breve descrizione;
  - Lista degli eventuali return, seguiti da una breve descrizione.

#### 4.2.5.2.3 Pratiche di programmazione

- **Metodi**: preferire più metodi semplici e con responsabilità singola rispetto a metodi lunghi e complessi;
- **Attributi**: mai accedere direttamente ad attributi di una classe. Utilizzare solo i metodi forniti per interagire con essa;
- **Self**: riferirsi ai parametri interni attraverso l'uso del riferimento self;
- ***Visibilità*$_{lgl}$**: usare la minima visibilità possibile per gli attributi;
- **Importazioni**: Importare i moduli all'inizio dello *script*$_{lgl}$ o del modulo. Evitare importazioni con asterisco (`from module import *`);
- **Commenti**: usare le seguenti convenzioni nei commenti:
  - *TODO*$_{lgl}$: per indicare sezioni di codice non ancora completate;
  - *FIXME*$_{lgl}$: per indicare sezioni di codice che necessitano di revisione e miglioramenti.

#### 4.2.5.2.4 Formattazione del codice

Per la formattazione del codice Python sono state adottate le convenzioni indicate dal *Formatter PEP8*$_{lgl}$ (`https://peps.python.org/pep-0008/` *(Ultimo accesso: 2024-02-08)*). Il rispetto della formattazione sopra indicata è garantito dallo strumento di formattazione incluso nell'estensione per il liguaggio Python di *VSCode*$_{lgl}$ fornita da Microsoft (`https://marketplace.visualstudio.com/items?itemName=ms-python.python` *(Ultimo accesso: 2024-02-08)*)

#### 4.2.5.2.5 Convenzioni sintattiche

- **Nomi autoesplicativi**: i nomi assegnati a qualsiasi costrutto devono essere autoesplicativi, cioè devono racchiudere il significato di ciò a cui si riferiscono; preferire nomi lunghi ma privi di ambiguità a nomi corti ma interpretabili;
- **Lingua**: usare la lingua inglese per i nomi dei costrutti e per i commenti;
- **Convenzioni di nomenclatura**: seguire le convenzioni di nomenclatura PEP 8;
- **Abbreviazioni**: usare cautamente le *abbreviazioni*$_{lgl}$, solo se queste sono comunemente conosciute;
- **Nomi classi**: i nomi delle classi devono seguire lo stile *PascalCase*$_{lgl}$;
- **Nomi metodi e variabili**: i nomi dei metodi e delle variabili devono seguire lo stile snake_case;
- **Nomi costanti**: i nomi delle costanti devono seguire lo stile *UPPER_ SNAKE_CASE*$_{lgl}$;
- **Nomi argomenti**: usare lo stesso nome per gli argomenti del costruttore e i rispettivi attributi della classe;
- ***Indentazione*$_{lgl}$**: utilizzare quattro spazi o un carattere di tabulazione per definire i blocchi di codice;
- ***Spaziatura*$_{lgl}$**: utilizzare uno spazio attorno agli operatori binari per una migliore leggibilità;
- **Parentesi**: aggiungere lo spazio prima e dopo un blocco raccolto tra parentesi;
- **Commenti**: utilizzare il simbolo # per i commenti;
- ***Quoting*$_{lgl}$**: utilizzare singoli (') apici per le stringhe;
- **Definizioni di funzioni e metodi**: separare le definizioni di funzioni e metodi con due linee vuote;
- **Lunghezza massima della linea**: limitare le linee a un massimo di 79 caratteri per il codice e 72 per docstrings;
- **Più istruzioni su una linea**: evitare più istruzioni sulla stessa linea.

### 4.2.5.3 Stile di codifica: Javascript (NextJS)

#### 4.2.5.3.1 Struttura dei file

La struttura di un sorgente *Javascript*$_{lgl}$ è definita dalle seguenti parti:

- Import: insieme di istruzioni di import che seguono l'ordine:
  1. Dipendenze esterne;
  2. Dipendenze interne.
- Funzioni: definiscono un componente o funzioni di utilità.

#### 4.2.5.3.2 Struttura dei componenti

La struttura di un *componente NextJS*$_{lgl}$ è definita dalle seguenti parti, se necessarie:

1. **Arrow function**: definisce il componente;

2. **Funzioni ausiliarie**: funzioni di supporto che aiutano a gestire casi complessi e aumentano il riuso;
3. **Logica del componente**: definisce la struttura e il comportamento del componente;
4. **Export**: istruzione di export che rende disponibile il componente all'utilizzo esterno.

Ogni classe deve essere opportunamente documentata, seguendo il formato *JSDoc*$_{|gl|}$; in particolare:

- Dopo la definizione del componente, devono essere presenti, in una serie di commenti:
  - Descrizione del ruolo della classe;
  - Lista dei *props*$_{|gl|}$, seguiti da una breve descrizione;
  - Lista dei metodi della classe, seguiti da una breve descrizione.
- Dopo la definizione dei metodi, devono essere presenti, in una serie di commenti:
  - Descrizione del metodo;
  - Lista di argomenti del metodo, seguiti da una breve descrizione;
  - Lista degli eventuali return, seguiti da una breve descrizione.

#### 4.2.5.3.3 Pratiche di programmazione

- **Grandezza componenti**: prediligere l'uso e la creazione di componenti di taglia e responsabilità limitata. Un componente dovrebbe limitarsi a svolgere un'unica funzione;
- **Riutilizzo**: sfruttare il più possibile il riutilizzo dei componenti. Se due componenti hanno la stessa struttura ma si differenziano per aspetto o comportamento, usare i props per gestire queste differenze;
- **Funzioni di utilità**: evitare di posizionare le funzioni di utilità all'interno di componenti, ma raccoglierle in classi specifiche e mirate;
- **Unico component per file**: esportare un unico component per file;
- **Metodi iterativi**: quando si vogliono eseguire operazioni su ogni elemento di un oggetto iterabile, prediligere l'uso di metodi iterativi;
- **Organizzazione file**: organizzare i file dello stesso component in una apposita cartella;
- **Commenti**: usare le seguenti convenzioni nei commenti:
  - TODO: per indicare sezioni di codice non ancora completate;
  - FIXME: per indicare sezioni di codice che necessitano di revisione e miglioramenti.

#### 4.2.5.3.4 Formattazione del codice

Per la formattazione del codice Javascript sono state adottate le convenzioni indicate dall'estensione *Prettier*$_{|gl|}$ di VSCode (`https://marketplace.visualstudio.com/items?`

`itemName=esbenp.prettier-vscode` *(Ultimo accesso: 2024-02-08)*). Per garantire il rispetto delle convenzioni di Prettier, tutti i componenti del gruppo hanno impostato il formatter Prettier di default e impostato il suo intervento correttivo ad ogni salvataggio dei file.

#### 4.2.5.3.5 Convenzioni sintattiche

- **Nomi autoesplicativi**: i nomi assegnati a qualsiasi costrutto devono essere autoesplicativi, cioè devono racchiudere il significato di ciò a cui si riferiscono; preferire nomi lunghi ma privi di ambiguità a nomi corti ma interpretabili;
- **Lingua**: usare la lingua inglese per i nomi dei costrutti e per i commenti;
- **Abbreviazioni**: usare cautamente le abbreviazioni, solo se queste sono comunemente conosciute;
- **Indentazione**: utilizzare quattro spazi o un carattere di tabulazione per definire i blocchi di codice;
- **Spaziatura**: utilizzare uno spazio attorno agli operatori binari per una migliore leggibilità;
- **Parentesi**: aggiungere lo spazio prima e dopo un blocco raccolto tra parentesi;
- **Quoting**: utilizzare singoli (`'`) apici per le stringhe;
- **Più istruzioni su una linea**: evitare più istruzioni sulla stessa linea;
- **Punto e virgola**: aggiungere sempre il simbolo ';' alla fine di ogni statement;
- **Nomi componenti**: i nomi dei componenti devono seguire lo stile PascalCase;
- **Nomi file componenti**: i nomi dei file che contengono un componente devono essere uguali al nome del componente;
- **Nomi *CSS*$_{lgl}$ componenti**: i nomi dei fogli di stile riferiti ad un unico componente devono essere uguali al nome del componente, ma scritti in *camelCase*$_{lgl}$;
- **Nomi cartelle componenti**: i nomi delle cartelle che contentengono un componente devono essere uguali al nome del componente, ma scritti in camelCase;
- **Nomi file ausiliari**: i nomi dei file ausiliari devono seguire lo stile camelCase;
- **Nomi metodi e variabili**: i nomi dei componenti devono seguire lo stile camelCase;
- **Nomi costanti**: i nomi delle costanti devono seguire lo stile UPPER_SNAKE_CASE.

### 4.2.5.4 Stile di codifica: HTML e CSS

Per lo *stile di codifica*$_{lgl}$ dei linguaggi *HTML*$_{lgl}$ e CSS è stato seguito il modello indicato al seguente indirizzo: `https://google.github.io/styleguide/htmlcssguide.html` *(Ultimo accesso: 2024-02-08)*.

### 4.2.5.5 Strumenti

- *VSCode*;
- *Github*.

#### 4.2.6 Testing del codice

In questa sezione vengono esposti i tipi e la notazione dei test che vengono eseguiti sul codice come parte del piano di verifica per garantire la correttezza del prodotto.

##### 4.2.6.1 Tipi di test

I test effettuati vengono suddivisi nelle seguenti categorie:

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

###### 4.2.6.1.1 Test di unità

- **Stesura**: i Verificatori sono incaricati di implementare i test di unità stabiliti durante la fase di progettazione di dettaglio, descritta in (§Progettazione di dettaglio);
- **Descrizione**: questi test sono progettati in conformità alle specifiche di ciascuna unità software, consentendo l'associazione di più test a una singola unità, qualora necessario, costruendo una suite di test dedicata a quell'unità. La loro esecuzione potrebbe richiedere l'utilizzo di stub o driver, strumenti che permettono di testare le singole unità simulando alcuni dei loro componenti, nel caso in cui non fossero tutti disponibili per l'esecuzione (pratica essenziale, specialmente quando l'ambiente è ancora piuttosto scarno);
- **Categorie**: i test di unità possono essere suddivisi in due categorie:
  - **Funzionali** (o **black box**): dato un input e un'aspettativa di output, verificano se l'esecuzione di una certa funzionalità produce l'aspettativa dichiarata;
  - **Strutturali** (o **white box**): verificano la logica interna del codice, analizzando i vari cammini di esecuzione all'interno dell'unità.

  I verificatori utilizzano entrambe queste tipologie di test di unità, sfruttando strumenti di automazione disponibili per le tecnologie adottate.

###### 4.2.6.1.2 Test di integrazione

- **Stesura**:i Verificatori sono incaricati di implementare i test di integrazione stabiliti durante la fase di progettazione di dettaglio, descritta in (§Progettazione di dettaglio);
- **Descrizione**: si applicano alle componenti individuate con lo scopo di rilevare difetti di progettazione, errori a livello di unit testing, incompatibilità e incongruenza nell'uso delle interfacce o errata integrazione con altre applicazioni. Devono assemblare incrementalmente, ampliando il loro raggio di azione di volta in volta;
- **Categorie**: esistono due strategie di integrazione:
  - **Bottom-up**: integrazione partendo dalle componenti con minor numero di dipendenze d'uso e maggiore utilità interna (molto chiamate ma che chiamano poco). Richiede pochi stub ma ritarda l'emissione di funzionalità visibili all'utente;

- **Top-down**: integrazione partendo dalle componenti con maggior numero di dipendenze d'uso e maggior utilità esterna (poco chiamate ma che chiamano molto). Comporta l'uso di molti stub ma integra prima le funzionalità più visibili all'utente.

#### 4.2.6.1.3 Test di sistema

- **Stesura**: questa fase di testing viene effettuata dai verificatori e avviene dopo il completamento dei test di unità e dei test di integrazione e prima del collaudo con il committente;
- **Descrizione**: i test di sistema mirano a valutare il sistema nel suo complesso per garantire che soddisfi tutti i requisiti delineati nel documento (Analisi dei requisiti v3.0.0(0)). Vegnono considerati come test funzionali (o black-box), non richiedono conoscenza della logica e del funzionamento interno del software.

#### 4.2.6.1.4 Test di regressione

- **Selezione**: questa fase di testing viene definita dai verificatori e avviene dopo il completamento dei test di unità, dei test di integrazione, dei test di sistema e prima del collaudo con il committente;
- **Descrizione**: i test di regressione sono un tipo specifico di test che include una selezione dei test già implementati (di unità, di integrazione e di sistema), per verificare che le modifiche apportate al software non abbiano introdotto nuovi difetti o non abbiano alterato il comportamento esistente del sistema. In altre parole, essi mirano a garantire che le modifiche apportate durante lo sviluppo del software non abbiano effetti collaterali negativi su funzionalità precedentemente testate e correttamente funzionanti.

### 4.2.6.2 Test: Notazione

I test vengono identificati con la seguente notazione: **T[Tipo].[Codice]**
nella quale:

- [**T**] indica la parola "test";
- [**Tipo**] può essere:
  - U (Unità);
  - I (Integrazione);
  - S (Sistema);
  - A (Accettazione).
- [**Codice**] identifica i test per ogni tipologia. È composto da un unico numero progressivo univoco se il test non ha padre, mentre se si tratta di un sotto-test, segue il formato **[Codice_padre].[Numero_figlio]**; questa struttura è ricorsiva, quindi non pone un limite alla profondità della gerarchia.

#### 4.2.6.3 Test: Stato

Ogni test è associato a uno stato che indica il risultato di soddisfacimento di esso nell'ultima versione rilasciata del prodotto software. Lo stato può assumere i seguenti valori:

- I (Indisponibile, test non in funzione);
- S (Superato);
- N (Non superato).

#### 4.2.6.4 Strumenti

- *VSCode*;
- *GitHub*.

### 4.2.7 Integrazione software

La *Continuous Integration*$_{|gl}$ (CI) è una pratica fondamentale nel nostro processo di sviluppo software che mira a integrare in modo continuo e automatico le modifiche al codice nel repository condiviso. Al fine di garantire la qualità e la stabilità del codice, oltre che permettere una panoramica immediata dello stato attuale dell'attività di codifica, è essenziale stabilire regole chiare riguardanti il processo di CI e il rilascio del software.
Di seguito sono elencate le norme di progetto relative alla CI.

#### 4.2.7.1 Branching

Il repository è strutturato in due branch principali:

- **Main**: branch principale del repository, contiene il codice sorgente stabile e funzionante;
- **PB**: branch di sviluppo, contiene il codice sorgente in fase di sviluppo e test relativi alla fase PB.

Ogni modifica o aggiunta di codice viene effettuata in un branch separato, creato a partire dal branch PB, seguendo la convenzione *'mod-code-Cognome'*, dove *'Cognome'* viene sostituito dal cognome del membro che ha creato il branch.

#### 4.2.7.2 Pull Request

Una volta completata la modifica o l'aggiunta di codice, il membro che ha creato il branch deve aprire una pull request per richiedere la revisione del codice.
Al fine di garantire un corretto versionamento (§Version control), il titolo della pull request deve seguire la convenzione:

- **Feat: *TEXT***: per l'aggiunta di nuove funzionalità, moduli o componenti, dove **TEXT** rappresenta una breve descrizione della funzionalità aggiunta;
- **Fix: *TEXT***: per la correzione di bug, errori o malfunzionamenti, dove **TEXT** rappresenta una breve descrizione del bug corretto.

Ogni apertura di una pull request comporta l'avvio automatico di una *Github Action*$_{lgl}$ denominata *'Test'*, che esegue i test di unità, di integrazione e sistema definiti nel documento (Piano di qualifica (v3.0.0(0)), §Test di unità, §Test di integrazione, §Test di sistema).
Grazie alla configurazione di adeguate Github Protection Rules attive nei branch *'main'* e *'PB'*, la pull request può essere approvata dai verificatori solamente a condizione di superamento di tutti i test presenti nel repository al momento della creazione della pull request.
Il team si impegna a garantire il corretto funzionamento dei test al momento della creazione della pull request, in modo da evitare l'introduzione di errori nel codice sorgente remoto, senza permettere la manipolazione dei test stessi al fine di ottenere l'approvazione della pull request.

#### 4.2.7.3 Strumenti

- *GitHub*;
- *GitHub Actions*.

# 5 Processi di supporto

Nella seguente sezione vengono presentati i processi di supporto allo sviluppo del progetto.

## 5.1 Documentazione

### 5.1.1 Scopo

Il processo di documentazione accompagna tutte le attività di sviluppo, con l'obiettivo di permettere una consultazione semplice e rapida alle informazioni inerenti al prodotto e alle attività stesse, svolgendo inoltre un ruolo di storicizzazione e di supporto alla manutenzione.

### 5.1.2 Implementazione del processo

In questa sezione viene documentato e implementato un piano che identifica i documenti da produrre durante il ciclo di vita del prodotto software. Tutti i documenti da redigere vengono presentati nella tabella che segue. Vengono inclusi solamente i documenti relativi al vero e proprio ciclo di vita del prodotto software e sono di conseguenza esclusi i documenti presentati per la candidatura per il progetto didattico, quali "Lettera di presentazione", "Preventivo costi e impegni orari" e "Valutazione dei capitolati d'appalto".

| **Nome del documento** | **Scopo** | **Redattore** | **Destinatari** | **Consegne previste** |
|---|---|---|---|---|
| Analisi dei requisiti | Definizione requisiti utente | Analista | Azienda proponente, Docenti | $RTB_{lgl}$, $PB_{lgl}$ |
| Norme di progetto | Definizione regolamento normativo per il team | Responsabile, Amministratore | Docenti | RTB, PB |
| Piano di progetto | Definizione temporale scadenze e progressi | Responsabile | Docenti | RTB, PB |
| Piano di qualifica | Definizione qualità e testing del prodotto finale | Amministratore | Docenti | RTB, PB |
| Verbali esterni | Tracciamento riunioni esterne | Responsabile | Azienda proponente, Docenti | Candidatura, RTB, PB |

| | | | | |
|---|---|---|---|---|
| Verbali interni | Tracciamento riunioni interne | Responsabile | Docenti | Candidatura, RTB, PB |

### 5.1.3 Progettazione e sviluppo

In questa sezione vengono presentati gli standard e le regole (nello specifico di stile) a cui i membri di SWEetCode si attengono relativamente alla stesura di documenti relativi al progetto.

#### 5.1.3.1 Template

Il team ha deciso di produrre un template grafico ricorrente all'interno della documentazione.
Ogni documento presenta la medesima pagina di copertina, con una grafica realizzata tramite *Adobe Illustrator* ed in seguito disposta nella pagina tramite *Overleaf*, dove viene esposto il nostro logo, il nostro nome del team e i membri del team.
Le seguenti pagine di ciascun documento presentano un:

- ***Header***: intestazione contenente il nostro logo, il nome del documento ed il nostro nome;
- ***Footer***: piè di pagina contenente i link alla nostra email e al nostro account *GitHub*$_{|gl|}$; ed infine la numerazione di pagina.

#### 5.1.3.2 Parametrizzazione template

Per facilitare e velocizzare la procedura di inserimento dei dati all'interno dei documenti prodotti con il linguaggio di marcatura *Latex*, il gruppo ha deciso di creare dei template parametrizzati, sfruttando l'uso di comandi già presenti nel linguaggio, come *\def* e *\newcommand*, e l'uso di librerie esterne, come *ifthen* e *forloop*. in questo modo la procedura di inserimento dei dati nei documenti avviene come segue:

- Il membro del gruppo che intende operare la modifica di un documento effettua la modifica locale del suo template (nel caso dei verbali) o del documento stesso (nel caso degli altri documenti);
- Durante la fase di compilazione la struttura dei documenti si adatterà in modo dinamico (considerando i valori dei parametri), assumendo l'aspetto del tipo di documento indicato, ad esempio verbale interno, esterno o altri tipi di documenti più discorsivi.

L'adozione di questa pratica permette al team di mantenere coerenti struttura, formato del contenuto e presentazione, aumentando il throughput e consentendo ai componenti del gruppo di concentrarsi più sul contenuto che sulla sua visualizzazione.

Di seguito vengono elencati i parametri utilizzati a seconda del tipo di documento:

- Titolo;
- Data;
- Orario inizio / fine;

- Luogo;
- Tipo di verbale (esterno o interno);
- Nome di responsabile, verificatore, segretario ed eventuale azienda;
- Firma di responsabile ed eventuale azienda;
- Lista dei partecipanti (interni e esterni);
- Lista revisione delle azioni;
- Lista ordine del giorno;
- Lista discussione (interna o esterna);
- Lista decisioni.

#### 5.1.3.3 Struttura di documento

Tutti i documenti prodotti da SWEetCode presentano una struttura comune, alla quale ogni membro si deve attenere durante le procedure di stesura e modifica della documentazione. Le procedure devono attenersi alla seguente struttura:

- **Pagina di copertina**: descritta nella sezione Template precedente;
- **Registro delle versioni**: presente in tutti i documenti ad eccezione dei verbali. Questo registro utilizzato per tenere traccia delle varie versioni permette di comprendere in modo rapido chi ha realizzato determinate sezioni della documentazione, quando sono state realizzate, cosa è stato aggiunto e perché (sotto la voce "Dettaglio e motivazioni"). Il registro presenta le versioni ordinate a partire dalla versione più recente;
- **Indice**: presente nel caso in cui si abbia un documento di notevole dimensione, dotato di sezioni. Il suo obiettivo è quello di facilitare e agevolare l'accesso ad un determinato contenuto trattato nel documento;
- **Contenuto**: il contenuto vero e proprio del documento.

#### 5.1.3.4 Verbali

Fanno eccezione alla struttura precedentemente esposta i verbali, sia esterni che interni. Essi infatti oltre a presentare nella prima pagina di copertina i ruoli inerenti alla loro produzione (al posto dell'elenco dei membri del team) ed una sezione apposita per la firma del responsabile di riunione (e in caso di verbale esterno anche la firma da parte di un rappresentate dell'azienda), offrono una struttura particolare.
Il loro contenuto presenta le seguenti sezioni:

- **Intestazione**: sezione contenente le informazioni inerenti a data, ora e luogo della riunione;
- **Partecipanti**: sezione contenente l'elenco in ordine alfabetico dei partecipanti interni ed eventualmente esterni alla riunione e una voce apposita per gli assenti;
- **Revisione delle azioni**: sezione contenente un elenco cronologico delle azioni assegnate nelle riunioni precedenti, con informazioni in merito al progresso, problematiche riscontrate ed eventuale conclusione di tale azione;

- **Ordine del giorno**: elenco cronologico e approssimativo delle tematiche trattate all'interno della riunione;
- **Discussione**: sezione contente l'elenco cronologico delle tematiche affrontate, trattate con una descrizione accurata delle informazioni emerse;
- **Decisioni**: decisioni prossime e/o future da intraprendere o intraprese a seguito della riunione affrontata;
- **Azioni da intraprendere**: tabella contente le azioni assegnate durante l'incontro e da svolgere nel futuro immediato. Ogni voce è accompagnata dal numero del *ticket* associato (se presente) nell'*ITS*$_{lgl}$ su *Jira*$_{lgl}$, un incaricato, da un revisore che dovrà verificare il suo operato e da una data di scadenza entro la quale l'azione da svolgere dovrà essere ritenuta completata;
- **Altro**: sezione contente informazioni aggiuntive quali riferimenti esterni e la data della prossima riunione (nel caso in cui essa sia stata stabilita).

#### 5.1.3.5 Posta elettronica

La posta elettronica rappresenta uno strumento fondamentale per le procedure di comunicazione asincrona con proponenti e committente.
Per uniformare lo stile delle mail, vengono descritti di seguito delle norme a cui attenersi nella loro stesura:

- **Oggetto**: campo da riempire obbligatoriamente in maniera breve e concisa;
- **Firma**: ogni mail viene conclusa con la firma digitale del gruppo collocata automaticamente;
- **Forma**:
  - Le informazioni e le richieste più importanti vanno posizionate all'inizio del messaggio;
  - La lunghezza del testo non deve essere eccessivo (onde evitare la perdita di visibilità di informazioni preziose);
  - Ci si deve servire dell'uso di paragrafi ed elenchi per strutturare il testo, qualora fosse complesso e denso di informazioni.
- **Stile**: è buona norma seguire lo stile del mittente. La mail sarà dotata di formule di saluto iniziale e finale quali ad esempio: *"Gentile.."*, *"Spettabile.."*, *"Cordiali saluti"* e *"Distinti saluti"*.

#### 5.1.3.6 Norme tipografiche

- **Citazioni tecnologiche e strumentali**: Ogni tecnologia e/o strumento menzionato all'interno della documentazione deve essere scritta in *corsivo*;
- **Date**: Tutte le date presenti nella documentazione prevedono il seguente formato: AAAA-MM-GG;
- **Elenchi**: Tutti gli elenchi presenti nella documentazione prevedono un ordine:
  - **Alfabetico**: sono puntati e non presentano alcuna correlazione con l'aspetto temporale;

- **Cronologico**: sono numerati e descrivono una serie di eventi o argomenti ordinati nel tempo.

In tutti gli elenchi ogni punto termina con il ";", fatta eccezione per l'ultimo elemento che termina con il ".";

- **Menzioni**: Ogni menzione di una persona, interna od esterna al team, avviene nel seguente formato: Cognome N. (dove la lettera N sta per l'iniziale del nome);
- **Nome del documento**: I nomi dei documenti prevedono la lettera iniziale maiuscola seguiti dal resto dei caratteri in case minuscolo e dalla versione del documento stesso.
  I verbali, sia interni che esterni, fanno eccezione e presentano un titolo composta dalla data in cui sono stati svolti nel seguente formato: AAAA-MM-GG;
- **Riferimenti interni**: Menzioni e riferimenti a sezioni interne allo stesso documento a cui appartengono vengono riportate seguendo la notazione (§Nome sezione). Questi riferimenti saranno opportunamente collegati tramite link al paragrafo indicato;
- **Riferimenti esterni**: Menzioni e riferimenti a sezioni di documenti esterni vengono riportate seguendo la notazione (Nome documento (versione di riferimento), §Nome sezione);
- **Nome del gruppo**: Il nome del gruppo presenta il seguente formato: "SWEetCode";
- **Stile tipografico**: La formattazione del testo dei documenti segue rigorosamente un unico stile tipografico: *Poppins*;
- **Versioni**: Ogni documento viene associato nel nostro sito GitHub alla versione del progetto per cui risulta essere ancora in vigore.
  All'interno del documento, nella prima riga del registro delle versioni, viene invece presentata la versione del progetto in cui è avvenuta l'ultima modifica al documento.
  L'incremento della versione del progetto a seguito di elaborazioni nella documentazione viene gestito attraverso la seguente norma: l'aggiunta di documentazione non precedentemente presente all'interno del repository o la modifica della documentazione precedentemente presente all'interno del repository porta all'incremento del solo valore "(build)".

### 5.1.4 Produzione

La produzione di ogni documento redatto da SWEetCode presenta le seguenti attività (numerate), eseguite in ordine cronologico (ogni attività viene a sua volta suddivisa in procedure minori):

1. **Assegnazione**: Attività svolta dal team ed articolata come segue:
   - Il documento viene assegnato ad uno o più responsabili di stesura affiancati a loro volta da uno o più revisori;
   - I responsabili di stesura discutono e decidono chi dovrà svolgere ogni parte del lavoro;

- Viene aperto su *Jira* un task associato ad ogni parte della realizzazione del documento;
- Il task appena creato viene immediatamente inserito all'interno di una epic e di una versione.

2. **Stesura**: La stesura viene realizzata dal segretario di riunione (nel caso in cui il documento in questione sia un verbale interno o esterno) o dai responsabili di stesura ed è operata come segue:
   - La produzione della documentazione in formato *Latex* viene gestita tramite la condivisione dei sorgenti attraverso *GitHub*, quindi è buona pratica che ogni membro ne possieda a priori la versione aggiornata. Se così non fosse, il responsabile di stesura aggiorna in locale la versione della documentazione, mediante un'istruzione *git pull*;
   - Il responsabile di stesura scarica il template necessario alla stesura del documento;
   - Il responsabile di stesura opera la stesura del documento;
   - Una volta che la stesura viene ritenuta stabile (dal responsabile di stesura), tramite l'automazione descritta in (Norme di progetto v3.0.0(0), §Automazione versionamento documenti), crea una *pull request* nel repository *Knowledge_Management_AI* su *GitHub*.
3. **Verifica e Validazione**: La fase di verifica viene realizzata dal revisore del documento e funziona come segue:
   - Il documento viene sottoposto ad un controllo di contenuto e sintassi da parte del verificatore, che tiene conto delle linee guida definite dal team nelle (Norme di progetto v3.0.0(0), §Documentazione);
   - Se il documento è di tipo esterno, una volta passata la verifica da parte del revisore, l'atto viene condiviso con l'ente terzo in causa, per essere sottoposto ad una sua ulteriore verifica e validazione.
   - In caso di esito positivo la *pull request* viene risolta mentre, in caso di esito negativo, essa viene rifiutata e si ritorna alla fase precedente, in modo da produrre una documentazione che tiene conto delle correzioni e precisazioni presentate dal revisore e/o dall'eventuale parte esterna.
4. **Pubblicazione**: La pubblicazione costituisce l'ultima fase del ciclo di vita del documento e avviene solamente nel caso in cui la fase di verifica abbia avuto un riscontro positivo, ed avviene come segue:
   - Una volta risolta la *pull request* il documento troverà collocazione all'interno del repository *Knowledge_Management_AI* del team su *GitHub*.

   In questo modo il repository contiene solamente documenti verificati ed eventualmente validati.

### 5.1.5 Manutenzione

Il processo di manutenzione della documentazione è fondamentale per garantire che la stessa rimanga accurata, aggiornata ed utile nel corso del ciclo di vita del progetto.

Tale processo è integrato in modo continuo in base ai progressi svolti. Il team segue le seguenti attività (suddivise in procedure) per il processo di manutenzione:

- **Identificazione della necessità della modifica**:
  - Monitoraggio continuo delle esigenze di modifica della documentazione (ogni membro opera il monitoraggio relativo ai documenti assegnati al ruolo che ricopre per ogni sprint);
  - Comunicazione al team della necessità di una possibile modifica.
- **Valutazione dell'impatto**:
  - Discussione tra i membri che ricoprono i ruoli interessati alla modifica per valutarne l'impatto sulla documentazione già esistente;
  - Nel caso si tratti di documenti molto onerosi o in caso di conflitti tra i membri che ricoprono i ruoli interessati, la dicussione viene estesa a tutto il team.
- **Aggiornamento della documentazione**:
  - Modifica e aggiornamento del documento interessato secondo le esigenze identificate;
  - Verifica dell'accuratezza delle informazioni e dell'allineamento con le modifiche del progetto relative.
- **Push della modifica**:
  - Per procedere, la stesura della modifica deve venire ritenuta stabile, tramite l'automazione descritta in (Norme di progetto v3.0.0(0), §Automazione versionamento documenti);
  - Viene creata una *pull request* nel repository *Knowledge_Management_AI* su *GitHub* per la nuova versione del documento da parte di chi ha operato la modifica.
- **Verifica e validazione**:
  - A seguito della creazione della *pull request*, le modifiche apportate ai documenti sono sottoposte a verifica eseguita da parte del verificatore;
  - Se tale operazione ha esito positivo, se necessario, il documento viene posto ad ulteriore validazione con l'ente esterno in questione;
  - Se sia la verifica che la eventuale validazione hanno avuto esito positivo, le modifiche vengono pubblicate attraverso l'accettazione della *pull request* da parte del revisore;
  - In caso invece di esito negativo in anche solo una delle due operazioni, la *pull request* viene rifiutata e colui che aveva realizzato le modifiche è responsabile di applicare le correzioni emerse, ripetendo i passi precedentemente descritti.

#### 5.1.6 Strumenti

- *Adobe Illustrator*;
- *GitHub*;

- *Latexmk(XeLatex)*;
- *Jira*.

## 5.2 Configuration management

### 5.2.1 Scopo

Nella seguente sezione vengono presentate le attività svolte dal gruppo SWEetCode che rientrano all'interno del processo "configuration management".

### 5.2.2 Descrizione

Il processo di configuration management è un processo di applicazione di procedure amministrative e tecniche lungo l'intero ciclo di vita del software al fine di:

- Identificare, definire e stabilire una base per gli elementi software in un sistema;
- Controllare le modifiche e le release degli elementi;
- Registrare e riportare lo stato degli elementi e delle richieste di modifica;
- Garantire la completezza, la coerenza e la correttezza degli elementi.

### 5.2.3 Configuration control

#### 5.2.3.1 Scopo

L'attività di configuration control si pone i seguenti obiettivi:

- Identificare e registrare le richieste di modifica;
- Analizzare e valutare le modifiche;
- Approvare o rifiutare le richieste di modifica;
- Tracciamento di audit.

#### 5.2.3.2 Issue tracking system (ITS)

Per raggiungere l'obiettivo di tracciamento di audit, ovvero la registrazione sistematica e dettagliata dei task e delle modifiche, SWEetCode utilizza l'*Issue Tracking System*$_{lgl}$ fornito da *Atlassian*: *Jira*.

##### 5.2.3.2.1 Ticket

In questo sistema di tracciamento si permette la creazione di *ticket* che possono essere di tre tipi:

- *task*: tale tipo di ticket viene utilizzato dal team per assegnate compiti atomici ai vari membri del gruppo;
- *bug*$_{lgl}$: tale tipo di ticket viene utilizzato dal team per assegnare un compito di fix di un bug all'interno del progetto ad un membro del gruppo;
- *story*: tale tipo di ticket viene utilizzato per rappresentare un requisito.

Ciascun *ticket* presenta (di seguito sono riportate le sole funzionalità utilizzate dal team):

- Titolo: un titolo breve ma esplicativo del compito associato al *ticket*;

- Id numerico univoco: tale id viene generato in automatico dall'ITS nella forma *SWE-ID*;
- *epic*: ogni *ticket* può essere infatti associato ad una *epic*;
- Collegamento: *Jira* mette a disposizione una funzionalità di collegamento per stabilire le dipendenze tra i vari *ticket*;
- Assegnatario: il membro del team che a cui è stata affidata la responsabilità di svolgere il *ticket*;
- Descrizione: una breve descrizione inerente al *ticket* associata;
- Data: la data di creazione del *ticket*;
- Reporter: il membro del team che ha creato il *ticket*;
- Sviluppo: voce contenente il link ai commit, alle pull request e alle build che sono stati effettuati nel repo su *GitHub* in correlazione al *ticket* in questione;
- Stato: gli stati possibili di un *ticket* sono *Da fare*, *In Progress*, *Pronto per commit* (introdotto dagli amministratori di SWEetCode nel flusso di lavoro) ed infine *Completato*;
- *Sprint*$_{lgl}$: ogni *ticket* può essere associato ad uno *sprint*;
- *Versione* di correlazione: ogni *ticket* può essere associato ad una determinata *versione*.

#### 5.2.3.2.2 Epic

Un insieme di *ticket* può essere raggruppato all'interno di una *epic*: una *epic* costituisce un'unità logica e strategica di lavoro più ampia, rappresentando un obiettivo o un risultato di alto livello. Questo strumento di lavoro offre al team una panoramica di completamento generale di un particolare insieme di azioni correlate tra loro, attraverso una scoreboard che indica la percentuale di *ticket* completati, in progresso o ancora da iniziare. Oltre ai *ticket* associati, ogni *versione* presenta i medesimi field di un *ticket* precedentemente descritti.

#### 5.2.3.2.3 Versioni

Per rappresentare le milestone su *Jira* vengono utilizzate le *Versioni*, alle quali possono essere associate *epic* e i relativi *ticket*, riportando come nelle *epic* una scoreboard che indica la percentuale di *ticket* completati, in progresso o ancora da iniziare. Una volta che tutti i *ticket* associati alla *versione* sono stati completati, tale *versione* può essere rilasciata. Oltre ai *ticket* associati, ogni *versione* presenta una breve descrizione, una data di inizio ed una di fine.

#### 5.2.3.2.4 Backlog e Sprint

*Jira* offre all'interno dell'*ITS* differenti funzionalità di supporto al metodo *Agile*$_{lgl}$, seguendo il framework *Scrum*$_{lgl}$, in particolare quando i *ticket* vengono creati sono inseriti di default all'interno del *Backlog* per essere in seguito assegnati ai vari *Sprint*. Ciascun *Sprint* è caratterizzato da una data di inizio ed una di fine e da uno stato (*Attivato*/*Chiuso*).

#### 5.2.3.2.5 Timeline

*Jira* mette a disposizione come strumento una *timeline* realizzata attraverso un *diagramma di Gantt*$_{|gl|}$ all'interno del quale si possono inserire i *ticket* creati associati ad una *epic*, ed il team la utilizza procedendo come segue:

- Ciascun *ticket* viene disposto in uno spazio temporale, facendo riferimento ad una data di inizio ed una di fine, costituendo il periodo all'interno del quale portare a termine il compito associato;
- Tra i vari *ticket*, se necessario, vengono creati dei collegamenti che rappresentano graficamente le dipendenze presenti tra essi;
- Ogni *versione* viene rappresentata all'interno della *timeline* da una linea verticale posta in corrispondenza della data di scadenza della versione stessa;
- Nella parte superiore della *timeline* sono visualizzati i vari *Sprint* presenti all'interno dell'ITS, visualizzando in opaco quelli chiusi;
- Internamente alla *timeline* è resa disponibile un'ulteriore funzionalità di filtraggio dei *ticket* visualizzati in base alle *versioni* e/o alle *epic*, che permette di effettuare un'istantanea.

#### 5.2.3.2.6 Automazione chiusura ticket

*Jira* mette a disposizione una funzionalità di creazione di automazioni (chiamate *Trigger*) configurabili dell'utente, permettendo di adattare al meglio ciascuna esigenza. Il team la sfrutta come segue:

- Si introduce un'automazione di chiusura automatica dei ticket presenti all'interno dell'*ITS*;
- Ogni volta che un membro del team effettua un commit nel repository finalizzato alla chiusura di un *ticket*, il contenuto di tale commit, grazie all'esecuzione di uno script Python creato dal team (Norme di progetto v3.0.0(0), Automazione versionamento documenti) presenterà al suo interno l'operazione "*SWE-#id_ticket*";
- A seguito dell'approvazione della pull request da parte del verificatore, il *ticket* associato a quel particolare *ID* passerà in automatico allo stato *Completato*.

### 5.2.3.3 Pull requests

A seguito dell'adozione dei *GitHub Teams*, SWEetCode propone l'utilizzo delle pull requests per permettere un rilascio controllato che permetta di analizzare e valutare le richieste di modifica effettuate: l'unico *Team* dotato dei permessi necessari all'approvazione di una pull request sono i *Verificatori*.

- Una volta proposta la pull request il verificatore attuale del progetto, membro del *Team Verificatori*, dovrà effettuare una verifica sulle modifiche proposte e, a seguito delle sue valutazioni, dovrà decidere se declinare o accettare la pull request, con conseguente merge nel branch all'interno del quale la pull request è stata sollevata in caso di esito positivo.

Tali responsabilità vengono definite tramite l'utilizzo di *Branch Protection Rules* offerte da *GitHub*. L'identificazione attraverso un titolo fornito da colui che solleva la pull re-

quest e la registrazione delle richieste di modifica e del loro esito è una conseguenza dell'applicazione della pratica delle pull request offerta da *GitHub*.

#### 5.2.3.4 GitHub Teams

SWEetCode, attraverso un profilo *Organizations*, ha deciso di utilizzare una struttura interna ai membri del gruppo che usufruisce dei *Teams* di *GitHub*: ogni ruolo presente all'interno del progetto porta alla creazione di un *Team* avente il medesimo nome.
I *Teams* presentati sono dunque:

- Amministratori;
- Analisti;
- Progettisti;
- Programmatori;
- Responsabili;
- Verificatori.

Lo strumento *Teams* consente di gestire i vari ruoli nel migliore dei modi, permettendo di assegnare responsabilità e permessi specifici a determinate posizioni piuttosto che ai membri del gruppo. Questa mansione altrimenti risulterebbe dispendiosa e disagevole a causa della rotazione dinamica interna dei ruoli.

### 5.2.4 Configuration status accounting

#### 5.2.4.1 Scopo

L'obiettivo che si pone il configuration status accounting è quello di mantenere una registrazione accurata e aggiornata del prodotto software e dei suoi elementi in relazione allo stato del prodotto stesso nel corso del tempo.

#### 5.2.4.2 Version control

Per tenere traccia dello stato e della storia degli elementi controllati, il team ha deciso di utilizzare un sistema di *Version control* basato su una versione del prodotto che si attiene al seguente formato: vX.Y.Z(build).

- v: sta per versione, rimane immutato lungo tutte le versioni;
- X: numero che indica la versione principale di riferimento. Viene incrementato quando viene superata una fase di revisione di avanzamento;
- Y: numero che viene incrementato con l'introduzione di nuove features o di miglioramenti significativi;
- Z: numero che indica piccoli cambiamenti rispetto alla versione vX.Y, come il fix di bug o il caricamento di nuovi documenti all'interno del repository (ad esclusione di verbali, sia interni che esterni);
- (build): numero che indica le build di progetto eseguite nella versione vX.Y.Z. Tale valore viene incrementato a seguito di modifiche/aggiunte nella documentazione.

Il sistema di numerazione delle versioni ha come prima versione il valore "v0.0.1(0)".
Ogni volta che viene incrementato il valore X, i valori Y, Z e (build) ripartono dai loro valori iniziali, ovvero "0".
Ogni volta che viene incrementato il valore Y, il valore Z e (build) ripartono dal valore "0".
Ogni volta che viene incrementato il valore Z, il valore (build) riparte dal valore "0".

### 5.2.5 Configuration evaluation

#### 5.2.5.1 Scopo

L'attività di configuration evaluation si pone l'obiettivo di garantire la completezza funzionale degli elementi software realizzati rispetto ai loro requisiti.

#### 5.2.5.2 Tracciamento dei requisiti

Per raggiungere l'obiettivo precedentemente citato, il team SWEetCode ha deciso di effettuare un tracciamento dei requisiti all'interno del prodotto software, in modo da portare nel concreto l'evidenza della correlazione tra requisiti e codice.
Per effettuare tale tracciamento il team apporta un commento, contenente l'identificativo del requisito, antecedente alla parte di codice che porta al soddisfacimento del requisito menzionato.

### 5.2.6 Release management

#### 5.2.6.1 Scopo

L'attività di release management si occupa di effettuare un rilascio controllato.

#### 5.2.6.2 Automazione release

L'automazione del *versionamento*$_{lgl}$ consente di rilasciare nuove release in modo automatico, seguendo le regole di cambio versione indicate in (Norme di progetto v3.0.0(0), §Version control) e avviene procedendo come segue:

- Attraverso l'uso di *GitHub Actions*$_{lgl}$, l'accettazione di una pull request, eseguita seguendo le norme indicate nel paragrafo (Norme di progetto v3.0.0(0), §Pull requests), è immediatamente seguita dalla pubblicazione della nuova versione (il tipo di cambio versione viene indicato dal prefisso del titolo della pull request).

Questa automazione si integra in modo efficace con quella citata nel paragrafo (Norme di progetto v3.0.0(0), §Automazione versionamento documenti). Ciò che separa l'esecuzione consecutiva delle due automazioni è la creazione della pull request e la sua accettazione. La richiesta dell'intervento umano, in questo caso, non è un fattore limitante, ma rappresenta due ulteriori fasi di verifica, che garantiscono la pulizia del repository remota e la correttezza del suo contenuto.

#### 5.2.6.3 Automazione versionamento documenti

L'aggiornamento del registro delle versioni, la creazione dei *PDF*$_{lgl}$ e la pubblicazione delle modifiche nel repository remoto avvengono in modo consecutivo e automatico grazie all'esecuzione di uno script Python creato dal team. Questo script richiede inizialmente l'inserimento di alcuni input riguardanti le modifiche, compresa l'eventuale

chiusura del *ticket* soddisfatto, e dopo una fase di elaborazione in cui aggiorna il registro delle versioni e crea il PDF usando il programma *Latexmk*, esegue il push in remoto dei cambiamenti introdotti in locale.
La procedura per l'aggiornamento è la seguente:

- Il componente del gruppo che carica o modifica un documento esegue il push in locale di tale documento, comilando lo script Python del team;
- Il componente del gruppo che ha eseguito lo script deve recarsi nel sito web del repository su GitHub e creare la Pull Request;
- Questa verrà successivamente analizzata e accettata o rifiutata in base ai criteri descritti nel (Piano di qualifica v3.0.0(0)) e secondo le modalità citate nel paragrafo (Norme di progetto v3.0.0(0), §Produzione).

L'utilizzo di questa automazione permette di ridurre gli errori legati all'aggiornamento e alla pubblicazione di nuove versioni dei documenti, e riduce notevolmente il carico di lavoro assegnato ai componenti del gruppo, limitando volutamente le loro azioni alla sola modifica del contenuto della documentazione.

## 5.3 Accertamento di qualità

### 5.3.1 Scopo

Lo scopo di questo processo è fornire un'adeguata garanzia che i prodotti software e i processi nel ciclo di vita del progetto siano conformi ai requisiti specificati e aderiscano ai loro piani stabiliti. L'assicurazione della qualità può utilizzare i risultati di altri processi di supporto, come la verifica e la validazione.

### 5.3.2 Process assurance

Per garantire che la qualità prefissata venga raggiunta e mantenuta, viene adottato il ciclo $PDCA_{lgl}$, noto anche come ciclo di Deming, un approccio metodologico utilizzato per il miglioramento continuo. Si divide in 4 fasi:

- **Plan**: questa fase coinvolge la definizione degli obiettivi di qualità, la pianificazione delle attività e la creazione di criteri di valutazione. Aiuta a stabilire le aspettative, i processi e le risorse necessarie per raggiungere gli standard di qualità prefissati;
- **Do**: durante questa fase, vengono implementate le azioni pianificate, eseguendo le attività identificate nel piano e raccogliendo dati per il monitoraggio della qualità;
- **Check**: questa fase consiste nella valutazione dei risultati ottenuti rispetto agli obiettivi prestabiliti. Attraverso metriche precise stabilite dal team, si verifica se i risultati sono in linea con le aspettative di qualità;
- **Act**: basandosi sull'analisi dei dati raccolti, si apportano correzioni migliorative ai processi esistenti, al fine di migliorare la qualità del prodotto.

### 5.3.3 Metriche: Notazione

Le metriche di qualità, definite successivamente, vengono identificate con la seguente notazione: **M.[Tipo].[Codice].[Sigla]**
nella quale:

- [**M**] indica la parola "metrica";
- [**Tipo**] indica il tipo di qualità:
  - **PC**: Qualità di processo;
  - **PD**: Qualità di prodotto.
- [**Codice**] è un numero progressivo univoco che identifica le metriche per ogni tipologia;
- [**Sigla**] composta dalle iniziali del nome della metrica; aiuta a identificarla più velocemente.

### 5.3.4 Metriche: Didascalia

Ogni metrica è descritta dai seguenti campi:

- **Nome**: nome della metrica;
- **Descrizione**: descrizione della metrica;
- **Caratteristica di riferimento**: caratteristica dello standard a cui la metrica fa riferimento;
- **Motivo**: motivazione per la quale è necessario misurarla;
- **Misurazione**: formula e strumenti tramite i quali viene calcolata.

### 5.3.5 Standard per la qualità del prodotto

Nella definizione delle metriche di qualità, il team si prefigge di seguire lo standard ISO/IEC 9126, che descrive un modello di qualità del prodotto dividendo la qualità in esterna, interna e in uso. Qualità esterna e interna sono organizzate attraverso le seguenti caratteristiche:

- **Funzionalità**: capacità del software di fornire le funzioni necessarie per operare in un determinato contesto;
- **Affidabilità**: capacità di un prodotto software di mantenere il livello di prestazione quando viene utilizzato in condizioni specifiche;
- **Usabilità**: capacità di un prodotto software di essere comprensibile. Chiarisce tutti gli ambienti e scenari di utilizzi del prodotto, inclusa la preparazione all'utilizzo del software e la valutazione dei risultati;
- **Efficienza**: capacità di un prodotto software di realizzare le funzioni richieste nel minor tempo possibile e utilizzando nel miglior modo le risorse necessarie;
- **Manutenibilità**: capacità di un prodotto di essere modificabile, sia per correzioni che per adattare il software e modifiche all'ambiente o ai requisiti;
- **Portabilità**: capacità di un prodotto di essere trasportato in ambienti diversi, con organizzazione e tecnologie diverse;

- **Qualità** in uso: rappresenta il punto di vista dell'utente sulla qualità interna ed esterna.

La qualità in uso viene descritta tramite le seguenti caratteristiche:

- ***Efficacia$_{lgl}$***: capacità di permettere all'utente di raggiungere obiettivi specifici con accuratezza e completezza in uno specifico contesto di utilizzo;
- **Produttività**: capacità di permettere all'utente di impegnare un numero definito di risorse, in relazione all'efficienza raggiunta in uno specifico contesto di utilizzo;
- **Sicurezza** fisica: capacità di raggiungere un livello accettabile di rischio per i dati, le persone, il business, la proprietà o gli ambienti;
- **Soddisfazione**: capacità di soddisfare gli utenti a seguito di interazioni con il prodotto.

### 5.3.6 Lista delle metriche

#### 5.3.6.1 Qualità del processo

- **M.PC.1.PMS**:
  - **Nome**: Percentuale di metriche soddisfatte;
  - **Descrizione**: Valore che indica la percentuale di metriche soddisfatte; una metrica si dice "soddisfatta" se la sua misurazione supera il valore accettabile specificato nel documento (Piano di qualifica v3.0.0(0));
  - **Caratteristica di riferimento**: -
  - **Motivo**: Questa metrica ci assicura il raggiungimento di un buon livello di qualità generale nello svolgimento del progetto;
  - **Misurazione**:
$$M = \frac{N°Metriche\ soddisfatte}{N°Metriche\ totali} * 100$$

- **M.PC.2.RNP**:
  - **Nome**: Rischi non previsti;
  - **Descrizione**: Valore che specifica il numero di rischi verificatisi che non rientrano tra quelli esaminati nel documento di (Analisi dei requisiti v3.0.0(0));
  - **Caratteristica di riferimento**: -
  - **Motivo**: Valore che indica se è stata fatta una accurata analisi dei rischi a monte della pianificazione.
  - **Misurazione**:
$$M = N°Rischi\ non\ previsti$$

- **M.PC.3.VP**:
  - **Nome**: Variazione di piano;

- **Descrizione**: Variazione utilizzata per analizzare se ogni attività o task del progetto, in un dato istante di tempo, è in linea o in ritardo rispetto alla pianificazione specificata nel documento (Piano di progetto v3.0.0(0)). Ai seguenti valori si associano i seguenti significati:
  - > 0 significa che si sta procedendo con un certo ritardo;
  - = 0 si è in linea con quanto pianificato.
- **Caratteristica di riferimento**: -
- **Motivo**: Valore che aiuta a controllare come il team procede e ad evitare ritardi;
- **Misurazione**:

$$M = TP - TC$$

  - TP= Numero di task pianificati;
  - TC= Numero di task completati.

- **M.PC.4.VC**:
  - **Nome**: Variazione di costo;
  - **Descrizione**: Valore che indica se il costo attuale del progetto è maggiore, uguale o minore rispetto al costo preventivato. Ai seguenti valori si associano i seguenti significati:
    - > 0 significa che si sta producendo con maggior efficienza risparmiando rispetto a quanto pianificato;
    - = 0 si è in linea con i costi pianificati;
    - < 0 significa che si sta producendo con minor efficienza causando maggiorazioni di costo rispetto a quanto pianificato.
  - **Caratteristica di riferimento**: -
  - **Motivo**: Valore che aiuta a controllare come si procede e ad evitare maggiorazioni di prezzo rispetto al preventivo;
  - **Misurazione**:

$$M = CP - CC$$

    - CP= Costo preventivato;
    - CC= Costo consuntivato.

- **M.PC.5.ISR**:
  - **Nome**: Indice di stabilità dei requisiti;

- **Descrizione**: Indice che traccia la variazione dei requisiti nell'arco del progetto;
- **Caratteristica di riferimento**: -
- **Motivo**: Questo valore indica eventuali discostamenti dai requisiti analizzati nel documento di (Analisi dei requisiti v3.0.0(0));
- **Misurazione**:

$$M = 100 - \frac{RM + RC + RA}{RTI} * 100$$

  - RM= Requisiti modificati;
  - RC= Requisiti cancellati;
  - RA= Requisiti aggiunti;
  - RTI= Requisiti totali iniziali.

- **M.PC.6.CCM**:
  - **Nome**: Complessità ciclomatica media;
  - **Descrizione**: Metrica utilizzata per valutare la complessità di un programma; misura direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso;
  - **Caratteristica di riferimento**: -
  - **Motivo**: Permette di controllare la complessità e di limitarla, semplificando il testing;
  - **Misurazione**:

$$M = \sum_{Metodo} \frac{e - n + 2}{NMT}$$

    - e= Numero di archi nel grafo dei cammini;
    - n= Numero di nodi nel grafo dei cammini;
    - NMT= Numero dei metodi totali.

- **M.PC.7.SC**:
  - **Nome**: Statement coverage;
  - **Descrizione**: Misura la percentuale di istruzioni nel codice eseguite dai test;
  - **Caratteristica di riferimento**: -
  - **Motivo**: Metrica che permette di valutare la correttezza del comportamento del software in esecuzione;
  - **Misurazione**:

$$M = \frac{N°Statement\ eseguiti}{N°Statement\ totali} * 100$$

- **M.PC.8.BC**:
  - **Nome**: Branch coverage;
  - **Descrizione**: Misura la percentuale di rami eseguiti o punti decisionali nel codice, come istruzioni *if* o *cicli*, e determina se i test esaminano sia il ramo vero che quello falso delle istruzioni condizionali;
  - **Caratteristica di riferimento**: -
  - **Motivo**: Metrica che permette di valutare la correttezza del comportamento del software in esecuzione;
  - **Misurazione**:
$$M = \frac{N°Branch\ eseguiti}{N°Branch\ totali} * 100$$

- **M.PC.9.ET**:
  - **Nome**: Efficienza temporale;
  - **Descrizione**: Percentuale che misura l'efficienza (in termini di ore impiegate) esercitata nello svolgere attività e task;
  - **Caratteristica di riferimento**: -
  - **Motivo**: Metrica che permette di valutare la produttività effettiva del team;
  - **Misurazione**:
$$ET = \frac{OR}{OP} * 100$$

    - OR= Numero di ore di orologio;
    - OP= Numero di ore produttive (ore che portano al raggiungimento di obiettivi).

#### 5.3.6.2 Qualità del prodotto

##### 5.3.6.2.1 Interne

- **M.PD.1.IG**:
  - **Nome**: *Indice di Gulpease*$_{Igl}$;
  - **Descrizione**: Questo indice descrive il grado di leggibilità di un testo tarato sulla lingua italiana. I valori sono compresi tra 0 (leggibilità più bassa) e 100 (leggibilità più alta). Ai seguenti valori si associano i seguenti significati:
    - < 80 sono difficili da leggere per chi ha la licenza elementare;
    - < 60 sono difficili da leggere per chi ha la licenza media;

    - < 40 sono difficili da leggere per chi ha un diploma superiore.
  - **Caratteristica di riferimento**: Usabilità-Comprensibilità;
  - **Motivo**: I documenti devono essere di facile fruizione per il maggior numero di persone possibili.
  - **Misurazione**:

$$M = 89 + \frac{300 * NF - 10 * NL}{NP}$$

    - NF= Numero delle frasi;
    - NL= Numero delle lettere;
    - NP= Numero delle parole.

- **M.PD.2.LMC**:
  - **Nome**: Linee medie di codice per metodo;
  - **Descrizione**: Valore rappresentante, in media, il numero di linee di codice che ogni metodo possiede;
  - **Caratteristica di riferimento**: Manutenibilità-Modificabilità;
  - **Motivo**: Questo valore serve a valutare la complessità del codice scritto;
  - **Misurazione**: Misurabile con strumenti di analisi statica.

- **M.PD.3.AFS**:
  - **Nome**: Adeguatezza delle funzioni sviluppate;
  - **Descrizione**: Metrica interna utilizzata per verificare in anticipo se il software soddisferà i requisiti funzionali;
  - **Caratteristica di riferimento**: Funzionalità-Adeguatezza;
  - **Motivo**: Misurare il livello di adeguatezza delle funzioni sviluppate rispetto alle esigenze specificate;
  - **Misurazione**:

$$M = \frac{FA}{FT}$$

    - **FA**= Numero di funzioni ritenute adeguate allo svolgimento del task specificato;
    - **FT**= Numero totale di funzioni sviluppate.

- **M.PD.4.AR**:
  - **Nome**: Accuratezza della risposta;

- **Descrizione**: Metrica che consente di misurare se i risultati ottenuti siano coretti, graditi e in linea con le aspettative;
- **Caratteristica di riferimento**: Funzionalità-Accuratezza;
- **Motivo**: Misurare il livello di adeguatezza delle risposte ottenute con la applicazione;
- **Misurazione**: Il team eseguirà varie strategie indicate in questa pagina web: `https://python.langchain.com/docs/guides/evaluation` *(Ultimo accesso: 2024-02-08)*.

- **M.PD.5.RD**:
  - **Nome**: Rimozione dei difetti;
  - **Descrizione**: Metrica interna di affidabilità che permette di valutare la maturità del prodotto dal punto di vista della riduzione delle possibili cause di malfunzionamenti;
  - **Caratteristica di riferimento**: Affidabilità-Maturità;
  - **Motivo**:Misura l'efficacia della rimozione degli errori rilevati nelle diverse fasi di sviluppo del prodotto;
  - **Misurazione**:

$$M = \frac{DC}{DT}$$

    - **DC**= Numero di difetti corretti;
    - **DT**= Numero totale di difetti rilevati nelle diverse fasi di sviluppo del prodotto (dati ricavati dai modelli statistici creati sulla base dei dati presenti nello storico del progetto).

- **M.PD.6.LCG**:
  - **Nome**: Livello di controllo dei guasti;
  - **Descrizione**: Metrica che aiuta a valutare la capacità del prodotto a mantenere le prestazioni desiderate anche in caso di malfunzionamenti;
  - **Caratteristica di riferimento**: Affidabilità-Tolleranza ai guasti;
  - **Motivo**: Misura il numero di condizioni di errore messe sotto controllo per evitare guasti al prodotto;
  - **Misurazione**:

$$M = \frac{E}{TC}$$

    - **E**= Numero di condizioni di errore gestite correttamente;

- **TC**= Numero totale di condizioni di errori possibili nel sistema.

- **M.PD.7.CD**:
  - **Nome**: Completezza descrittiva;
  - **Descrizione**: Metrica che valuta se dei nuovi utenti possono comprendere i seguenti punti:
    - Se il software è adatto alle loro esigenze;
    - Come il software possa essere utilizzato per particolari task.
  - **Caratteristica di riferimento**: Usabilità-Comprensibilità;
  - **Motivo**: Misurare il livello di completezza delle funzionalità descritte nella documentazione del prodotto;
  - **Misurazione**:

$$M = \frac{D}{FT}$$

    - **D**= Numero di funzioni descritte nella documentazione;
    - **FT**= Numero totale di funzioni previste.

- **M.PD.8.IM**:
  - **Nome**: Impatto delle modifiche;
  - **Descrizione**: Metrica che permette di valutare quanto stabile risulta il prodotto software dopo aver apportato modifiche;
  - **Caratteristica di riferimento**: Manutenibilità-Stabilità;
  - **Motivo**: Misurare l'impatto negativo sulla corretta esecuzione del software procurato dalle modifiche del codice;
  - **Misurazione**:

$$M = \frac{E}{MT}$$

    - **E**= Numero di modifiche che hanno procurato un malfunzionamento del software o hanno influito negativamente sulle prestazioni;
    - **MT**= Numero totale di modifiche eseguite.

#### 5.3.6.2.2 Esterne

- **M.PD.9.TS**:
  - **Nome**: Percentuale di test superati;
  - **Descrizione**: Valore che sta a indicare la percentuale di test che si concludono con successo;

- **Caratteristica di riferimento**: Funzionabilità-Accuratezza;
- **Motivo**: Rappresenta la capacità di un prodotto software di fornire i risultati o gli effetti attesi con il livello di precisione attesa;
- **Misurazione**:

$$M = \frac{N°Test\ superati}{N°Test\ totali} * 100$$

- **M.PD.10.CSD**:
  - **Nome**: Correttezza dello scambio di dati (Percezione degli utenti);
  - **Descrizione**: Metrica per misurare la mancanza di problemi nelle funzioni e strutture dati richieste per lo scambio di dati e di comandi tra il prodotto ed altre applicazioni con cui quest'ultimo interagisce ed è connesso;
  - **Caratteristica di riferimento**: Funzionabilità-Interoperabilità;
  - **Motivo**: Misura il livello di correttezza dei dati scambiati con altri applicativi così come percepito dagli utenti;
  - **Misurazione**:

$$M = 1 - \frac{SF}{NT}$$

    - **SF**= Numero di volte in cui il prodotto fallisce nello scambiare dati con altri applicativi;
    - **NT**= Numero totale di volte in cui l'utente cerca di scambiare dati.

- **M.PD.11.AFPH**:
  - **Nome**: Accesso fisico alle funzioni da parte di personale portatore di handicap;
  - **Descrizione**: Serve per misurare se gli utenti diversamente abili siano in grado di operare con il prodotto;
  - **Caratteristica di riferimento**: Usabilità-Operabilità;
  - **Motivo**: Misura il numero di funzioni disponibili del prodotto accessibili da parte di portatori di handicap;
  - **Misurazione**:

$$M = \frac{N°funzioni\ accessibili}{N°funzioni\ totali}$$

- **M.PD.12.TR**:
  - **Nome**: Tempo di risposta;
  - **Descrizione**: Fa parte delle metriche esterne di comportamento rispetto al tempo necessario, serve per misurare l'efficienza del prodotto;

- **Caratteristica di riferimento**: Efficienza-Comportamento rispetto al tempo;
- **Motivo**: Misura il tempo medio in cui il sistema risponde ad un comando immesso dall'utente;
- **Misurazione**:

$$M = T$$

  - **T**= Tempo che intercorre tra l'immissione del comando da parte dell'operatore e la presentazione della risposta del sistema.

- **M.PD.13.EI**:
  - **Nome**: Efficienza dell'installazione;
  - **Descrizione**: Metrica che serve per misurare il lavoro richiesto ad un operatore che debba installare il software in un determinato ambiente;
  - **Caratteristica di riferimento**: Portabilità-Installabilità;
  - **Motivo**: Misura il tempo necessario ad installare e personalizzare il prodotto nel nuovo ambiente;
  - **Misurazione**:

$$M = \frac{OI}{OT}$$

    - **OI**= Numero di ore impiegate per installare e personalizzare il prodotto nel nuovo ambiente;
    - **OT**= Numero di ore previste dal design e dalle specifiche.

### 5.3.7 Obiettivi di qualità: Struttura

Gli obiettivi di qualità, divisi in obiettivi di processo e di prodotto, vengono indicati nel documento (Piano di qualifica v3.0.0(0)). Ognuno di questi è descritto da:

- **Metrica**: codice della metrica;
- **Nome**: nome esteso della metrica;
- **Valore tollerabile**: valore oltre il quale la metrica è da considerare soddisfatta;
- **Valore ambito**: valore ideale della metrica.

### 5.3.8 Strumenti

- *Google Sheet*;
- *Jira*.

## 5.4 Verifica

### 5.4.1 Scopo

Il processo di verifica ha lo scopo di determinare se ciò che viene prodotto è conforme ai requisiti stabiliti e ai vincoli qualitativi specificati nel (Piano di qualifica v3.0.0(0)). Lo scopo principale della verifica è garantire la completezza e la correttezza del prodotto, fornendo evidenze oggettive e misurabili.

### 5.4.2 Descrizione

Ogni produzione realizzata da SWEetCode viene sottoposta ad uno o più fasi di verifica prima di essere pubblicata ufficialmente nel repository *GitHub*. Questo processo viene svolto dai verificatori. Onde evitare conflitti di interesse, tale compito viene assegnato ad uno o più membri diversi da coloro che hanno conseguito alla realizzazione del prodotto in questione.

### 5.4.3 Analisi statica

L'analisi statica è un'attività di verifica che costituisce un'esplorazione dell'oggetto di verifica senza richiedere l'esecuzione di esso. Può essere estesa anche ai documenti testuali.
Due tecniche di analisi statica utilizzate dal team, il walkthrough e l'ispezione, sono descritte di seguito.

#### 5.4.3.1 Walkthrough

Il walkthrough è un'approfondita procedura collaborativa in cui il verificatore e il realizzatore del prodotto leggono criticamente tutto l'oggetto in esame. In particolare, questa attività è composta dalle seguenti fasi:

- **Pianificazione**: coinvolge sia il verificatore che l'autore. In questa fase vengono definite le modalità, le risorse e le tempistiche entro le quali svolgere l'attività;
- **Lettura**: il verificatore esamina l'intero prodotto per individuare errori, incongruenze e migliorie;
- **Discussione**: il verificatore comunica all'autore tutti gli elementi individuati e ritenuti migliorabili; in questa sessione vengono definite anche possibili soluzioni e correzioni;
- **Correzione dei difetti**: Una volta accordata la modalità e il contenuto delle modifiche, l'autore procede con l'effettiva attuazione delle correzioni.

#### 5.4.3.2 Ispezione

L'ispezione è una tecnica che utilizza liste di controllo per eseguire verifiche mirate sugli errori più comuni, evitando la lettura dell'intero documento. L'ispezione è composta dalle seguenti fasi:

- **Pianificazione**: coinvolge sia il verificatore che l'autore. In questa fase vengono definite le modalità, le risorse e le tempistiche entro le quali svolgere l'attività;

- **Definizione lista di controllo**: vengono definiti i punti e gli aspetti critici dell'oggetto in esame; queste liste vengono costruite incrementalmente e aggiornate nel momento in cui la frequente occorrenza di certi errori diventa evidente;
- **Lettura**: il verificatore esamina il prodotto seguendo la lista di controllo per individuare errori, incongruenze e migliorie;
- **Correzione dei difetti**: L'autore procede con l'attuazione delle correzioni.

Data l'iniziale inesperienza del team, per i primi periodi è stato utilizzato l'approccio walkthrough. Nel momento in cui le liste di controllo diventeranno sufficientemente ampie per permettere una verifica adeguata, l'ispezione permetterà al team di svolgere questa attività più velocemente, preservando una grande quantità di risorse.
Le liste di controllo sono raccolte nel documento (Piano di qualifica v3.0.0(0), §Checklist).

#### 5.4.4 Analisi dinamica

L'analisi dinamica è una metodologia di verifica che richiede l'esecuzione effettiva del software al fine di valutarne il comportamento, le performance e la correttezza durante l'esecuzione. Questo tipo di analisi deve poter essere automatizzata e ripetibile. Per questo motivo, devono essere stabilite e regolamentate le condizioni dell'ambiente di esecuzione dei test. L'analisi dinamica viene resa possibile dall'implementazione di varie categorie di test, in particolare: di unità, di integrazione, di sistema e di regressione;
Le descrizioni e classificazioni di questi test vengono esplicitate nella sezione (§Testing del codice).

### 5.5 Validazione

#### 5.5.1 Scopo

Il processo di validazione è finalizzato a determinare se i requisiti stabiliti da contratto all'interno del documento (Analisi dei requisiti v3.0.0(0)) vengono soddisfatti nel prodotto sottoposto a tale processo.

#### 5.5.2 Implementazione del processo

La validazione viene effettuata in presenza dell'azienda proponente.
Precondizione necessaria dell'avvio di questo processo è l'esito positivo dei:

- Test di unità;
- Test di integrità;
- Test di sistema.

Lo svolgimento di tale processo coinvolge i seguenti passi:

1. **Tracciamento dei requisiti**: il team espone all'azienda proponente il tracciamento dei requisiti realizzato, portando così l'evidenza della presenza dei requisiti all'interno del prodotto posto a validazione;
2. **Collaudo**: il team, assieme all'azienda proponente, esegue il prodotto. L'esecuzione del prodotto viene accompagnata dall'esecuzione dei test di accettazione.

Se i test di accettazione effettuati durante il collaudo terminano con esito positivo, il prodotto realizzato dal team viene ritenuto validato dall'azienda proponente.

### 5.5.3 Test di accettazione

- **Stesura**: questi test vengono definiti in concomitanza con l'analisi dei requisiti;
- **Descrizione**: i test di accettazione prevedono la simulazione degli scenari principali di utilizzo del prodotto da parte degli utenti.
  Questi test permettono di testare le principali funzionalità che vengono rese disponsibili dal prodotto sottposto a validazione, portando ad una rapida evidenza del soddisfacimento di alcuni dei requisiti richiesti, compatibile coi tempi di un collaudo (la responsabilità di dimostrare la completezza del prodotto nei confronti di tutti requisiti richiesti da contratto viene infatti affidata al tracciamento dei requisiti).

## 5.6 Revisione congiunta

### 5.6.1 Scopo

La revisione congiunta consiste nel definire le attività per valutare lo stato e i prodotti di un'attività. Questo processo può essere utilizzato da qualsiasi coppia di parti, in cui una parte (la parte che esegue la revisione) valuta un'altra parte (la parte che viene revisionata) in un forum congiunto.

### 5.6.2 Implementazione del processo

La revisione congiunta avviene tra il team SWEetCode e l'azienda proponente attraverso un incontro accordato da entrambe le parti tramite *Google Calendar*, tenuto attraverso la piattaforma *Google Meet*.
L'implementazione di tale processo prevede il seguente svolgimento:

1. Definizione del materiale da sottoporre a revisione;
2. Definizione di una data e un orario per svolgere l'incontro tra le due parti;
3. Svolgimento dell'incontro tra le parti in cui il team espone i progressi e lo stato del materiale da sottoporre a revisione all'azienda proponente;
4. Documentazione tramite verbale esterno dei risultati della revisione congiunta e degli eventuali problemi rilevati.

### 5.6.3 Revisioni di project management congiunte

Le revisioni di project management congiunte pongono a revisione i seguenti materiali:

- Stato del progetto in relazione alle pianificazioni presenti nel (Piano di progetto v3.0.0(0));
- Scadenze valutate sulla base dello stato del prodotto in relazioni alle date di consegna accordate.

I fini di tale revisione congiunta consistono nel far progredire le attività secondo quanto pianificato, mantenendo il controllo del progetto e permettendo di cambiare direzione o definire nuove necessità ove si presentasse l'esigenza nel modo più responsivo

possibile: attraverso ciò il team ritiene di ridurre il rischio di divergere dalle aspettative dell'azienda proponente nei confronti del servizio offerto.

### 5.6.4 Revisioni tecniche congiunte

Le revisioni tecniche congiunte pongono a revisione i prodotti software e la documentazione, considerando la completezza e la conformità alle specifiche.
Tale tipo di revisione permette di apportare modifiche correttive concordate. Questo tipo di revisione congiunta consente di acquisire feedback correttivi tempestivi, agevolando un rapido aggiustamento delle eventuali lacune o discrepanze. Inoltre, tali revisioni forniscono un importante mezzo per valutare se il team stia procedendo nella direzione corretta, contribuendo così a mantenere il progetto allineato agli obiettivi prefissati.

### 5.6.5 Strumenti

- *Google Calendar*;
- *Google Meet*.

## 5.7 Revisione

### 5.7.1 Scopo

Il processo di revisione è un processo interno al team per determinare la conformità ai requisiti, ai piani e al contratto.

### 5.7.2 Implementazione del processo

La revisione avviene all'interno del team SWEetCode e prevede il seguente svolgimento:

1. Definizione del materiale da sottoporre a revisione, con assegnazione di uno o più revisori, attraverso il supporto fornito dall'ITS *Jira*;
2. Revisione completa da parte dei verificatori una volta ritenuto ultimato il materiale in questione;
3. Definizione di una data e un orario per svolgere l'incontro tra i membri del team;
4. Svolgimento dell'incontro attraverso la piattaforma *Discord*$_{|gl|}$, in cui i membri che hanno realizzato il materiale da sottoporre a revisione espongono rapidamente i relativi progressi ed il relativo stato del materiale ai restanti membri del team, seguiti dalle considerazioni effettuate dai revisori;
5. Documentazione tramite verbale interno dei risultati della revisione e degli eventuali problemi rilevati.

### 5.7.3 Revisioni di project management

Le revisioni di project management pongono a revisione i seguenti materiali:

- Stato del progetto in relazione alle pianificazioni presenti nel (Piano di progetto v3.0.0(0));

- Scadenze valutate sulla base dello stato del prodotto in relazioni alle date di consegna accordate.

I fini di tale revisione consistono nel far progredire le attività secondo quanto pianificato, anche in termini di tempi e costi, mantenendo il controllo del progetto e permettendo di cambiare direzione o definire nuove necessità ove si presentasse l'esigenza nel modo più responsivo possibile: attraverso ciò il team ritiene di individuare quelle che sono le criticità e i punti di debolezza del way of working adottato, in modo da attuare e, dove possibile, rimpiazzare tali aspetti per offrire un servizio di maggior qualità.

### 5.7.4 Revisioni tecniche

Le revisioni tecniche pongono a revisione i prodotti software e la documentazione, considerando la completezza e la conformità alle specifiche e agli standard.
Tale tipo di revisione permette di apportare modifiche correttive attraverso feedback interni, agevolando un rapido aggiustamento delle eventuali lacune o deviazioni dalle aspettative di produzione del team nei confronti del lavoro prodotto. Inoltre, tali revisioni forniscono un importante mezzo per valutare lo stato di avanzamento dei prodotti software e della documentazione, contribuendo così a mantenere il progetto allineato agli obiettivi prefissati.

### 5.7.5 Strumenti

- *Discord*;
- *Jira*.

## 5.8 Risoluzione dei problemi

### 5.8.1 Scopo

Il processo di risoluzione dei problemi analizza e risolve problemi di qualsiasi natura o sorgente, rilevati durante l'esecuzione dei processi primari. Il suo obiettivo è fornire un servizio tempestivo e documentato per garantire che tutti i problemi rilevati siano analizzati e risolti, riconoscendo le cause e adottando soluzioni che ne evitano la ricomparsa, qualora fosse possibile.

### 5.8.2 Implementazione del processo

1. Nel momento in cui viene rilevato un problema in un prodotto software o in un'attività, ogni componente del gruppo è tenuto a segnalarlo tempestivamente, tracciandolo nell'ITS *Jira* come ticket di tipologia Bug e informando le parti coinvolte della sua esistenza attraverso i canali di comunicazione descritti nella sezione Comunicazione;
2. Chi ha rilevato il bug redige un rapporto di problema per descrivere ciascun problema rilevato (inclusi non conformità);
3. Il rapporto di problema deve essere utilizzato come parte del processo a ciclo chiuso: rilevazione del problema -> indagine e analisi -> risoluzione del problema e della sua causa -> rilevazione delle tendenze dei problemi:

- Per categorizzare e prioritizzare i problemi, ogni problema viene classificato per **categoria** e **priorità**, in modo da facilitare l'analisi e la risoluzione, attraverso i labels e il tag **Urgency** messi a disposizione dal ticket Bug dell'ITS;
- Il segnalatore, oltre a tracciare il problema, deve anche fornire una descrizione dettagliata dello stesso, indicando le **condizioni** per le quali il problema si verifica e le eventuali **cause** individuate.

4. Una volta segnalato un problema, le parti coinvolte devono azionarsi per risolverlo: in base alla descrizione fornita dal segnalatore, i risolutori analizzano il problema e cercano una soluzione valida;
5. Se ciò non basta, i risolutori sono tenuti ad entrare in contatto con il segnalatore, collaborando e ottenendo altre informazioni utili a riguardo;
6. Ogni attività di risoluzione dei problemi viene verificata e valutata, per garantire che le loro attuazioni siano state correttamente implementate e non abbiano introdotto altri problemi.

### 5.8.3 Strumenti

- *Jira*.

# 6 Processi organizzativi

## 6.1 Gestione organizzativa

### 6.1.1 Scopo

Lo scopo di questo processo è esporre le modalità e gli strumenti di coordinamento usati dal gruppo per la comunicazione interna ed esterna e normare l'assegnazione dei ruoli e dei compiti e la gestione dei rischi.

### 6.1.2 Ruoli

Al fine di ottimizzare la gestione delle diverse attività e compiti da svolgere, il progetto definisce sei distinti ruoli, ciascuno con mansioni e responsabilità specifiche. Ogni componente del gruppo dovrà assumere ciascun ruolo per un numero di ore significativo. Di seguito vengono elencati i ruoli, seguiti da una breve descrizione.

#### 6.1.2.1 Responsabile

Il responsabile oltre ad essere il punto di riferimento per tutto il gruppo, è anche il punto di riferimento per le comunicazioni con il committente e con l'azienda proponente. Oltre a ciò, il responsabile è la figura all'interno del gruppo che ha il compito di coordinare le azioni dei vari membri del team. Per questo motivo, deve avere competenze tecniche in ogni ambito del progetto. Le sue responsabilità riguardano:

- La pianificazione;
- La gestione dei task e delle risorse;
- Il coordinamento tra gruppo e enti esterni;
- La gestione del dialogo interno;
- La gestione dell'avanzamento del progetto;
- L'approvazione di proposte introdotte da membri del gruppo.

#### 6.1.2.2 Amministratore

L'amministratore definisce, controlla, e manutiene l'ambiente di sviluppo. Si occupa della gestione della configurazione, del versionamento, delle varie automazioni e della documentazione. Ha il compito di:

- Selezionare e abilitare risorse informatiche a supporto del way of working;
- Gestire errori e segnalazioni di malfunzionamenti di meccanismi nell'infrastruttura.

#### 6.1.2.3 Analista

Questo ruolo ha la funzione di analizzare il problema e definire i requisiti del prodotto. Necessita di una buona conoscenza del dominio del problema. Raccoglie le sue produzioni nel documento (Analisi dei requisiti v3.0.0(0)). Questo ruolo è fondamentale all'inizio, ma la sua utilità cala con l'avanzare del progetto.

#### 6.1.2.4 Progettista

Si occupa di determinare le scelte realizzative e le specifiche architetturali del prodotto. Deve avere buone competenze tecniche e tecnologiche. L'utilità di questo ruolo è alta inizialmente, durante il processo di sviluppo, ma tende a calare dalla fase di manutenzione in poi.

#### 6.1.2.5 Programmatore

Contribuisce alla realizzazione effettiva del prodotto. In particolare:

- Si occupa di codificare ciò che è stato definito dai progettisti;
- Implementa i test;
- Redige il Manuale utente.

#### 6.1.2.6 Verificatore

Ha il compito di verificare il lavoro svolto dagli altri. Ha profonde competenze tecniche ed è presente durante l'intera durata del progetto. In particolare verifica che ciò che viene prodotto sia conforme alle norme e alle aspettative di qualità definite dal team.

### 6.1.3 Definizione delle attività

Ogni membro del team può proporre attività da svolgere durante la fase di pianificazione, ma sarà poi compito del responsabile stabilirne la fattibilità rispetto alle risorse usufruibili all'interno dello sprint.

### 6.1.4 Pianificazione delle attività

Le attività, dopo essere state definite, verranno pianificate nel tempo e nelle risorse dal responsabile. Il responsabile valuterà e deciderà quindi:

- La tempistica oraria necessaria per il completamento dell'attività;
- Il membro che dovrà realizzare l'attività, in base al ruolo e alle risorse disponibili;
- Il verificatore;
- Il rischio associato, attraverso le metriche.

Dopo di ciò, l'amministratore avrà il compito di inserire la task associata su *Jira* compilando tutti i campi richiesti:

- Assegnatario;
- Sprint;
- Stima originaria;
- Urgency.

Tale task dovrà inoltre essere collocato nello spazio temporale del *diagramma di Gantt* utilizzato dal team, sempre presente all'interno dell'ITS *Jira*.
Molte di queste procedure sono documentante più nel dettaglio nella sezione (§Configuration management).

#### 6.1.4.1 Strumenti

- *Jira*.

### 6.1.5 Esecuzione delle attività

L'esecuzione dell'attività avverrà da parte dell'assegnatario, definito dal responsabile. Tale esecuzione dovrà essere obbligatoriamente conforme alla documentazione associata, la cui stesura deve essere antecedente all'esecuzione dell'attività. L'esecutore dell'attività dovrà proporre una soluzione effettuando una pull request.

### 6.1.6 Revisione delle attività

La revisione dell'attività viene effettuata dal verificatore prima dell'effettiva introduzione delle modifiche nel repository *GitHub*: la pull request sollevata dall'esecutore viene infatti accettata o rifiutata a seconda dell'esito della verifica effettuata.

#### 6.1.6.1 Strumenti

- *GitHub*.

### 6.1.7 Chiusura delle attività

In caso di esito positivo:

- Viene accettata la pull request;
- Viene chiuso il branch di cui è stato fatto il merge;
- L'attività viene contrassegnata automaticamente come completata, grazie alle automazioni indicate nella sezione (§Configuration management).

Nel caso in cui il verificatore non ritenga corretto o sufficiente il lavoro svolto, o nel caso in cui la verifica automatica abbia esito negativo, il verificatore dovrà indicare tramite review le parti non valide e gli accorgimenti che si potrebbero apportare. L'esecutore dell'attività dovrà ripresentare una nuova soluzione seguendo le correzioni indicate dal verificatore.

### 6.1.8 Tracciamento orario

Avviene secondo le seguenti modalità:

- Il team utilizza un *Google Sheet* condiviso ai fini di tener traccia della quantità di tempo utilizzato per svolgere le attività;
- Ogni componente del gruppo è tenuto a registrare in questo foglio, dopo ogni sessione lavorativa, il numero di ore effettive della sua durata, oltre al ruolo svolto.

#### 6.1.8.1 Strumenti

- *Google Sheet*.

### 6.1.9 Comunicazione

SWEetCode ha deciso di mantenere una comunicazione frequente tra i membri del gruppo. In particolare, di seguito vengono esposte le modalità di comunicazione, interna ed esterna, e le applicazioni utilizzate.

#### 6.1.9.1 Comunicazioni interne

##### 6.1.9.1.1 Comunicazioni sincrone

Il gruppo ha deciso che gli incontri si svolgeranno in presenza oppure attraverso la piattaforma *Discord*.

- **Incontri in presenza**: verranno decisi alcuni giorni prima in modo tale da poter consentire la presenza di tutti i membri del gruppo;
- ***Discord***: su questa piattaforma si opta per una comunicazione asincrona anche pochi minuti prima del collegamento.

In entrambe le situazioni, il gruppo utilizza un approccio libero alla discussione e improntato alla crescita.

##### 6.1.9.1.2 Comunicazioni asincrone

Il gruppo utilizzerà *Whatsapp* e *Notion* per le comunicazioni asincrone:

- ***Whatsapp***: in una *community* sono stati creati:
  - Un gruppo principale per comunicazioni informali;
  - Una bacheca nella quale verranno inviati i messaggi più importanti dei quali è necessario prendere visione;
  - Altri canali in cui i componenti del gruppo si organizzano per task collaborativi.

##### 6.1.9.1.3 Strumenti

- *Discord*;
- *Whatsapp*.

#### 6.1.9.2 Comunicazioni esterne

##### 6.1.9.2.1 Comunicazioni sincrone

- **Incontri da remoto**: tali incontri sono cruciali per discutere con maggiore facilità e immediatezza argomenti complessi ed estesi:
  - Il gruppo ha concordato ed accettato, con l'azienda proponente, un calendario di incontri settimanali o bisettimanali su piattaforma *Google Meet*;
  - Il gruppo si impegna a presenziare in maniera più assidua possibile agli incontri con il proponente e a redarre un verbale al fine di documentare gli stessi.

##### 6.1.9.2.2 Comunicazioni asincrone

Per le comunicazioni asincrone con il proponente/soggetti esterni vengono utilizzati:

- ***Slack***: canale scelto e concordato con l'azienda; usato per brevi comunicazioni e per accordarsi sugli orari degli incontri;

- **Posta elettronica**: per comunicare con soggetti esterni diversi dall'azienda proponente, verrà utilizzata sempre e solo l'indirizzo mail ufficiale del gruppo: **sweetcode.team@gmail.com**.

#### 6.1.9.2.3 Strumenti

- *Google Gmail*;
- *Google Meet*;
- *Slack*.

### 6.1.9.3 Moderazione

La gestione della comunicazione viene affidata al responsabile. In particolare, ha il compito di:

- Fissare la data, l'orario e il luogo delle riunioni interne;
- Stabilire gli argomenti di discussione e l'ordine del giorno;
- Agire da moderatore della discussione, garantendo pari opportunità di espressione ad ogni componente del gruppo;
- Comunicare con l'esterno, dopo aver preso delle decisioni con il team.

### 6.1.9.4 Norme comportamentali

Al fine di garantire il rispetto delle norme e mantenere rapporti sereni all'interno del gruppo, ogni componente ha l'obbligo di:

- Essere puntuale o eventualmente comunicare tempestivamente al responsabile eventuali indisposizioni;
- Partecipare attivamente ad ogni discussione;
- Mantenere un comportamento rispettoso e disciplinato, aperto a discussioni e cambi di posizione.

### 6.1.9.5 Gestione dei rischi

La definizione e la gestione dei rischi sono attività svolte dal responsabile e dall'amministratore. I risultati di queste attività sono incluse nel documento (Piano di progetto v3.0.0(0)). Le seguenti sezioni definiscono le modalità tramite le quali questi rischi vengono documentati.

#### 6.1.9.5.1 Notazione

I rischi sono indicati con la notazione seguente:
**R[Tipo].[Numero]** nella quale:

- **R** sta per *Rischio*;
- **Tipo** può assumere i seguenti valori:
  - **P** (Personale);
  - **O** (Organizzativo);

- **T** (Tecnologico).

- **Numero** è un numero univoco progressivo, correlato ad ogni tipo, che identifica il rischio.

#### 6.1.9.5.2 Didascalia

La definizione dei rischi è dettagliata nel seguente modo:

- **Nome**: nome del rischio;
- **Descrizione**: descrizione del rischio;
- **Occorrenza**: misura di occorrenza del rischio (Alta, Media, Bassa);
- **Impatto**: misura di impatto del rischio (Alto, Medio, Basso);
- **Misure di mitigazione**: descrizione delle misure da adottare in caso di occorrenza della situazione problematica descritta dal rischio.

## 6.2 Infrastruttura

### 6.2.1 Scopo

Il processo di infrastruttura ha lo scopo di definire e documentare l'infrastruttura utilizzata dal team e il modo in cui questa viene stabilita e manutenuta.

#### 6.2.1.1 Implementazione dell'infrastruttura

Questa attività definisce e documenta l'infrastruttura e le modalità di installazione e configurazione della stessa. Il ruolo che si occupa dello svolgimento di questa attività è l'amministratore; egli ha ha il compito di implementare le scelte infrastrutturali.

1. L'amministratore procede con la verifica della fattibilità delle scelte considerando i seguenti aspetti:
    - **Funzionalità**: le scelte fatte devono essere sempre migliorative, aggiungendo o migliorando funzionalità. Il team deve ragionare a fondo su ogni scelta, cercando di evitare l'introduzione di malfunzionamenti o comportamenti indesiderati;
    - **Performance**: una modifica dell'infrastruttura deve categoricamente portare ad un miglioramento o al mantenimento delle performance precedenti;
    - **Sicurezza**: le scelte fatte non devono andare a impattare negativamente la sicurezza dell'infrastruttura;
    - **Disponibilità**: la configurazione deve essere disponibile e attuabile per ogni dispositivo utilizzato nel progetto. Onde evitare discrepanze tra infrastrutture implementate, la disponibilità deve essere totale;
    - **Vincoli di spazio**: l'infrastruttura deve rispettare i vincoli di spazio imposti dai dispositivi dei componenti;
    - **Attrezzatura disponibile**: la configurazione scelta deve essere compatibile con le configurazioni hardware a disposizione dei componenti del gruppo;

- **Costi**: In particolare, vengono preferite soluzioni open-source o gratuite rispetto a opzioni a pagamento, anche se queste dovessero rappresentare la scelta migliore;
- **Vincoli di tempo**: le fasi di installazione e configurazione dell'infrastruttura devono essere limitate in termini di tempo, evitando di rallentare l'avanzamento del progetto. Per ovviare a ciò, si preferisce posticipare le modifiche migliorative non obbligatorie, destinandole a periodi meno densi.

2. Se questa fase di verifica risulta soddisfatta, l'amministratore procede nel documentare le procedure necessarie per l'installazione e la configurazione, considerando anche le divergenze che possono nascere dalla diversità di dispositivi utilizzati dai membri; queste vengono raccolte in una sezione apposita del workspace *Notion*;
3. Una volta finita la fase di documentazione, l'amministratore informa i componenti del team, che procederanno nel seguire e attuare i passaggi documentati;
4. Nell'eventualità in cui alcuni componenti riscontrassero problemi durante le fasi di installazione e configurazione, l'amministratore ha il compito di seguirli passo per passo, individuando e risolvendo i problemi il prima possibile:
   - Nel caso in cui questi passaggi fossero particolarmente complicati e critici, il responsabile, in accordo con l'amministratore, organizzerà degli incontri in cui verranno spiegate in dettaglio le procedure necessarie.

#### 6.2.1.2 Manutenzione dell'infrastruttura

- L'infrastruttura deve essere mantenuta, monitorata e modificata dall'amministratore, in modo che continui a soddisfare le aspettative;
- Il funzionamento e lo stato dell'infrastruttura vanno verificati periodicamente;
- L'amministratore ha il compito di intervenire tempestivamente in presenza di difetti o malfuzionamenti, fornendo una correzione definitiva, o quantomeno temporanea ma risolutiva;
- I membri del gruppo sono tenuti a segnalare eventuali guasti, proporre migliorie e rispettare l'infrastruttura, collaborando durante le fasi di configurazione e tenendosi aggiornati riguardo le novità introdotte.

#### 6.2.1.3 Strumenti

- *Notion*.

## 6.3 Miglioramento

### 6.3.1 Scopo

Il processo di miglioramento è un processo che stabilisce, misura e controlla il miglioramento per tutto il ciclo di vita del software.

### 6.3.2 Determinazione del processo

Il team stabilisce una serie di processi del ciclo di vita del software sulla base dello *Standard ISO/IEC 12207:1997*: tali processi e la loro applicazione devono essere docu-

mentati e, dove possibile, viene istituito un meccanismo di controllo del processo per monitorare, controllare e migliorare il processo in questione.

### 6.3.3 Valutazione del processo

L'attività di valutazione del processo avviene attraverso le procedure seguenti:

- Il team attraverso revisioni interne, revisioni congiunte e procedure di valutazione del processo, elabora un giudizio;
- Lo storico di tali valutazioni viene conservato attraverso la documentazione e attraverso i report offerti da *Jira*, i quali determinano le esigenze di avanzamento;
- Il team deve quindi pianificare ed eseguire revisioni dei processi secondo intervalli appropriati per garantirne continuità, adeguatezza ed efficacia alla luce dei risultati della valutazione.

### 6.3.4 Miglioramento del processo

Il team deve apportare miglioramenti ai suoi processi come determinato necessario a seguito della valutazione e della revisione dei processi, procedendo come segue:

- La documentazione del processo viene aggiornata per riflettere i miglioramenti introdotti, evidenziando i punti di forza e di debolezza introdotti.

## 6.4 Formazione

### 6.4.1 Scopo

Il processo di formazione mira a normare le modalità di formazione dei componenti del gruppo per quanto riguarda le tecnologie necessarie per la produzione della documentazione e la realizzazione del prodotto software.

### 6.4.2 Piano di formazione

- Ogni componente del gruppo è libero di formarsi nel modo che ritiene più opportuno;
- Ogni componente del gruppo sostiene il processo di apprendimento 'learning by doing';
- Il gruppo sostiene che lasciare questa libertà possa ampliare la visione di gruppo riguardo le tecnologie utilizzate, aumentando la consapevolezza e migliorando complessivamente le scelte implementative;
- Verranno organizzati dei workshop (successivamente documentati) in cui i membri più esperti informeranno gli altri componenti e chiariranno eventuali dubbi per alleviare discrepanze di conoscenza.

### 6.4.3 Raccolta materiale di formazione

Il materiale di formazione utilizzato dal gruppo viene raccolto in una sezione apposita del workspace *Notion*. Queste fonti vengono organizzate per tecnologia e aggiornate di comune accordo.

#### 6.4.3.1 Strumenti

- *Notion*.

# 7 Strumenti

Nella seguente sezione vengono presentati tutti gli strumenti utili al team per lo svolgimento di ogni attività.

## 7.1 Adobe Illustrator

https://www.adobe.com *(Ultimo accesso: 2024-02-08)*

Software utilizzato per la creazione dei template ufficiali del team, per la generazione del logo e per la scelta della palette di colori da usare.

## 7.2 Canva

https://www.canva.com *(Ultimo accesso: 2024-02-08)*

Piattaforma utilizzata per la produzione delle slide per i diari di bordo.

## 7.3 Diagrams.net

https://app.diagrams.net/ *(Ultimo accesso: 2024-02-08)*

Applicazione utilizzata per la creazione dei diagrammi UML.

## 7.4 Discord

https://discord.com *(Ultimo accesso: 2024-02-08)*

Piattaforma utilizzata per lo svolgimento di riunioni formali ed informali del team, attraverso la creazione di un server apposito con un canale dedicato ai messaggi di testo e diverse sale dedicate alle conversazioni vocali.

## 7.5 GitHub

https://github.com/ *(Ultimo accesso: 2024-02-08)*

Piattaforma di hosting e collaborazione per lo sviluppo di software che offre strumenti per il controllo di versione, consentendo agli sviluppatori di lavorare insieme, monitorare le modifiche al codice, gestire problemi e pubblicare il proprio lavoro in modo collaborativo.

## 7.6 Google Calendar

https://calendar.google.com *(Ultimo accesso: 2024-02-08)*

Applicazione di calendario basata su cloud sviluppata da Google che consente agli utenti di organizzare gli eventi, le riunioni e le attività quotidiane, sincronizzando automaticamente le informazioni su tutti i dispositivi connessi e facilitando la condivisione di calendari con altri utenti.

### 7.7 Google Gmail

https://mail.google.com *(Ultimo accesso: 2024-02-08)*

Servizio di posta elettronica sviluppato da Google che offre una piattaforma di email, con funzionalità avanzate di organizzazione.

### 7.8 Google Sheet

https://docs.google.com/spreadsheets *(Ultimo accesso: 2024-02-08)*

Applicazione di fogli elettronici basata su cloud sviluppata da Google che consente agli utenti di creare, modificare e condividere fogli di calcolo online, facilitando la collaborazione in tempo reale, offrendo funzionalità avanzate di analisi dati, formule e grafici.

### 7.9 Latexmk(XeLatex)

https://pypi.org/project/latexmk.py/ *(Ultimo accesso: 2024-02-08)* Strumento di automazione utilizzato per semplificare il processo di compilazione dei documenti scritti in LaTeX utilizzando il motore di composizione XeLaTeX. Automatizza la sequenza di passaggi necessari per produrre un documento finale, gestendo automaticamente le dipendenze e garantendo che la compilazione avvenga in modo efficiente.

### 7.10 Jira (di *Atlassian*)

https://www.atlassian.com/it/software/jira *(Ultimo accesso: 2024-02-08)*

Software utilizzato per l'ITS e gli strumenti di organizzazione del lavoro tra i membri del team.

### 7.11 Notion

https://www.notion.so/product *(Ultimo accesso: 2024-02-08)*

Spazio di lavoro utilizzato per l'organizzazione delle prime task divise per membro del gruppo e per tenere traccia delle attività e dei progressi portati avanti da ogni componente. Usato anche per l'assegnazione chiara dei ruoli, come calendario condiviso del gruppo e per il salvataggio link utili in modo da favorire l'accesso in maniera veloce a tutti i documenti ed i siti necessari.

### 7.12 Slack

https://slack.com *(Ultimo accesso: 2024-02-08)*

Piattaforma di messaggistica specializzato nella comunicazione e collaborazione all'interno di team aziendali. Viene utilizzata dal team per le comunicazioni asincrone con il proponente.

### 7.13 VSCode

https://code.visualstudio.com *(Ultimo accesso: 2024-04-09)*

Ambiente di sviluppo integrato (IDE) utilizzato per la scrittura del codice sorgente, la gestione dei repository e la compilazione del codice.

### 7.14 Whatsapp

https://www.whatsapp.com *(Ultimo accesso: 2024-02-08)*

Applicazione di messaggistica, che rende disponibile le *Community*, una funzionalità dedicata ai gruppi che permette di creare sottogruppi tematici all'interno di un gruppo principale, facilitando la comunicazione e l'organizzazione all'interno di un team.