Norme di Progetto

Gruppo Argo — ***Progetto ChatSQL***

Informazioni sul documento	
Versione	0.0.12
Approvazione	TODO
Uso	Interno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo Argo

UNIVERSITÀ DEGLI STUDI DI PADOVA

Registro delle modifiche

Ver.	Data	Redazione	Verifica	Descrizione
0.0.12	2024-06-11	Riccardo Cavalli	Martina Dall'Amico	Aggiunta sezione pull request
0.0.11	2024-06-10	Riccardo Cavalli	Martina Dall'Amico	Completata sezione dashboard *Google Sheets*$_G$
0.0.10	2024-06-10	Riccardo Cavalli	Martina Dall'Amico	Formalizzazione della struttura del repository documentale
0.0.9	2024-06-03	Raul Pianon	Marco Cristo, Riccardo Cavalli	Modifica sezione *Analisi dei Requisiti*
0.0.8	2024-06-03	Sebastiano Lewental	Riccardo Cavalli, Raul Pianon, Marco Cristo	Aggiornamento metriche
0.0.7	2024-05-18	Martina Dall'Amico	Sebastiano Lewental	Descrizione metriche
0.0.6	2024-05-06	Riccardo Cavalli	Tommaso Stocco	Automazione chiusura *ticket*$_G$
0.0.5	2024-05-06	Riccardo Cavalli	Tommaso Stocco	*Ticket*$_G$ su Jira e integrazione *Jira*$_G$/*GitHub*$_G$
0.0.4	2024-05-04	Riccardo Cavalli	Martina Dall'Amico	Formalizzazione della struttura dei verbali e aggiornamento della tabella Todo dopo il passaggio a *Jira Software*$_G$
0.0.3	2024-04-28	Riccardo Cavalli	Martina Dall'Amico	Dashboard *Google Sheets*$_G$
0.0.2	2024-04-26	Riccardo Cavalli	Martina Dall'Amico, Mattia Zecchinato	Convenzioni per l'uso delle lettere maiuscole

Continua nella prossima pagina

Ver.	Data	Redazione	Verifica	Descrizione
0.0.1	2024-04-20	Tommaso Stocco	Martina Dall'Amico, Mattia Zecchinato	Creazione e stesura iniziale del documento

Indice

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di delineare le *best practices*$_G$ e il *way of working*$_G$ che il gruppo *Argo* ha individuato e adotta durante tutto lo svolgimento del progetto didattico. Poiché il way of working è definito incrementalmente durante il corso del progetto, questo documento non è da considerarsi un testo definitivo o completo.

1.2 Scopo del prodotto

1.3 Glossario

Allo scopo di evitare incomprensioni relative al linguaggio utilizzato nella documentazione di progetto, viene fornito un *Glossario*, nel quale ciascun termine è corredato da una spiegazione che mira a disambiguare il suo significato. I termini tecnici, gli acronimi e i vocaboli ritenuti ambigui vengono formattati in corsivo all'interno dei rispettivi documenti e marcati con una lettera $_G$ in pedice. Tutte le ricorrenze di un termine definito nel *Glossario* subiscono la formattazione sopracitata.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- C9 ChatSQL: creare frasi SQL da linguaggio naturale (Zucchetti S.p.A.):
  `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9.pdf`
  `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9p.pdf`;
- Standard ISO/IEC 12207:1995:
  `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf`;

1.4.2 Riferimenti informativi

- Documentazione ufficiale di GitHub:
  https://docs.github.com
  (Ultimo accesso: 2024-06-12);
- Documentazione ufficiale di Git:
  https://www.git-scm.com
  (Ultimo accesso: 2024-05-15);
- I workflow di Git a confronto:
  https://www.atlassian.com/git/tutorials/comparing-workflows
  (Ultimo accesso: 2024-06-10);
- Documentazione ufficiale di Jira:
  https://confluence.atlassian.com/jira
  (Ultimo accesso: 2024-05-15);

- Continuous integration:
  https://it.wikipedia.org/wiki/Integrazione_continua
  (Ultimo accesso: 2024-05-15);
- Continuous delivery:
  https://www.atlassian.com/it/continuous-delivery
  (Ultimo accesso: 2024-05-15);
- Slide T2 - Corso di Ingegneria del Software - Processi di ciclo di vita del Software:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T2.pdf
  (Ultimo accesso: 2024-04-11);
- Slide T3 - Corso di Ingegneria del Software - Modelli di sviluppo del Software:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T3.pdf
  (Ultimo accesso: 2024-04-11);
- Slide T4 - Corso di Ingegneria del Software - Gestione di Progetto:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T4.pdf
  (Ultimo accesso: 2024-04-11);
- Normative e standard:
  https://www.humanwareonline.com/project-management/center/differenze-tra-normative-standard
  (Ultimo accesso: 2024-05-02);
- *Piano di Progetto*;
- *Piano di Qualifica*;
- *Analisi dei Requisiti*;
- Verbali interni ed esterni.

2 Processi primari

2.1 Fornitura

2.1.1 Descrizione

Il *processo*$_G$ di fornitura consiste nell'insieme di attività e compiti svolte dal *Fornitore*$_G$ nel rapporto con la *Proponente*$_G$ Zucchetti S.p.A.. Il processo parte dalla candidatura al *capitolato*$_G$ d'appalto e prosegue con la determinazione di procedure e risorse richieste per la gestione e assicurazione del progetto, incluso lo sviluppo e l'esecuzione di un *Piano di Progetto*$_G$. L'obiettivo principale del processo è confrontare le aspettative della Proponente con i risultati del Fornitore durante il periodo del progetto, mantenendo dunque una metrica oggettiva tra il preventivato e lo stato corrente.
Il processo consiste nelle seguenti attività:

- Selezione e studio fattibilità;
- Candidatura;
- Pianificazione;

- Esecuzione e controllo;
- Revisione e valutazione;
- Consegna e completamento.

2.1.1.1 Selezione e studio fattibilità Il Fornitore esamina i capitolati d'appalto e arriva a una decisione sulla candidatura per uno di essi.

2.1.1.2 Candidatura Il Fornitore definisce e prepara una candidatura al capitolato d'appalto scelto producendo i seguenti documenti:

- **Lettera di Candidatura:** presentazione del gruppo rivolta al *Committente*$_G$;
- **Stima dei Costi e Assunzione Impegni:** documento che contiene un preventivo sulla distribuzione ore del progetto, il suo costo, una pianificazione generale e una iniziale analisi dei rischi;
- **Valutazione Capitolati:** documento che contiene l'analisi e la valutazione da parte del gruppo dei capitolati disponibili.

2.1.1.3 Pianificazione Il Fornitore stabilisce i requisiti per la gestione, lo svolgimento e la misurazione della qualità del progetto. In seguito, sviluppa e documenta attraverso il *Piano di Progetto*$_G$ i risultati attesi.

2.1.1.4 Esecuzione e controllo Il Fornitore esegue il Piano di Progetto sviluppato, attenendosi alle norme definite nella sezione 2.2 e monitora la qualità del *prodotto software*$_G$ nei seguenti modi:

- Controllo del progresso di *performance*$_G$, costi, rendicontazione dello stato del progetto e organizzazione;
- Identificazione, tracciamento, analisi e risoluzione dei problemi.

2.1.1.5 Revisione e valutazione Il Fornitore coordina la revisione interna ed esegue verifica e validazione secondo le norme definite in 4 e 5. Questo avviene in modo continuo e iterativo.

2.1.2 Rapporti con la Proponente

TODO

2.1.3 *Documentazione*$_G$ fornita

Di seguito viene descritta la documentazione che il gruppo si impegna a rendere disponibile alla Proponente e ai Committenti.

2.1.3.1 Piano di Progetto TODO

2.1.3.2 Analisi dei Requisiti

- La struttura di un UC è divisa nella seguente maniera:
  - UCN - Nome UC;
  - Descrizione;
  - Precondizioni;
  - Postcondizioni;
  - Scenario principale;
  - Eventuale Scenario alternativo;
  - Eventuale Inclusioni;
  - Eventuali Estensioni.
- Le precondizioni dello UC devono essere condizioni necessarie per arrivare alla situazione che si presenta nello UC. Ad esempio per l'utilizzo dello strumento di debug per il Tecnico, saranno:
  - Il Tecnico ha effettuato il login;
  - Il Tecnico ha ricevuto un prompt generato sulla base di una richiesta in liguaggio naturale;
  - Il Tecnico decide di utilizzare lo strumento di debug.
- Le precondizioni vengono poste al passato per identificare la conclusione di un UC;
- Le postcondizioni rappresentano cosa succede dopo lo sviluppo dello UC e sono per tanto descrittive e poste al presente;
- Lo scenario principale riprende i passaggi che sono stati necessari per il verificarsi dello UC. Pertanto si parte a descriverle dalla prima estensione che l'attore riscontra dopo l'avvio dell'applicativo fino ad arrivare allo UC.
- Lo scenario alternativo riprende i passaggi dello scenario principale fino all'estensione che porta allo UC alternativo.
- Ogni riferimento ad un UC viene riportato in precondizioni, postcondizioni, scenario principale, scenario alternativo, inclusione ed estensione.
- I sottocasi d'uso di uno UC vengono inseriti sotto lo UC principale nello stesso file.
- I sottocasi sempre inclusioni del caso d'uso "padre".
- I sottocasi vengono riferiti con un punto dopo il padre. Ad esempio un sottocaso potrebbe essere UC.1 e il sottocaso del sottocaso UC.1.1
- Per gli errori si visualizza quasi sempre un messaggio. Quindi la postcondizione finale e la fine dello scenario principale sarà quasi sempre "Viene visualizzato un messaggio con i dettagli dell'errore".

2.1.3.3 Piano di Qualifica TODO

2.1.3.4 Lettera di Presentazione TODO

2.1.3.5 Glossario Raccolta esaustiva di tutti i termini tecnici utilizzati nella documentazione. Permette di eliminare ambiguità e fraintendimenti fornendo una defizione univoca ed esaustiva per l'intero gruppo e per chi consulta la documentazione prodotta.

2.1.4 Strumenti

Gli strumenti impiegati nel processo di fornitura sono:

- **LaTeX:** *markup language*$_G$ utilizzato per la redazione della documentazione;
- **Git:** *Version Control System*$_G$ adottato dal gruppo;
- **Zoom:** Piattaforma per le riunioni con la Proponente e i Committenti;
- **Google Sheets:** Strumento per la creazione di *spreadsheet*$_G$ condivisi, utilizzato per la pianificazione di *sprint*$_G$ e per la rendicontazione delle ore;
- **Google Moduli:** Strumento per la creazione di questionari di valutazione degli *sprint*$_G$;
- **Table Convert Online:** Strumento online per convertire file *csv*$_G$ in *LaTeX*$_G$ (disponibile al seguente link https://tableconvert.com/it/csv-to-latex);
- **PDF24 Tools:** Strumento online per convertire file SVG in PDF (disponibile al seguente link https://tools.pdf24.org/it/svg-in-pdf).

2.1.4.1 Dashboard Google Sheets

Lo *spreadsheet*$_G$ condiviso, realizzato tramite *Google Sheets*$_G$ e visibile su *Google Drive*$_G$, ha lo scopo di automatizzare la generazione di preventivi e consuntivi (orari ed economici). Il foglio di calcolo principale è suddiviso nei seguenti fogli interni:

- Un foglio contenente le variabili globali del *Piano di Progetto*, inclusi il costo orario di ciascun ruolo e il budget totale;
- Un foglio nascosto per ciascun periodo, che include le risposte al questionario di valutazione dello sprint;
- Un foglio per ogni sprint con le seguenti informazioni:
  - Numero dello *sprint*$_G$;
  - Date di inizio e termine dello *sprint*$_G$;
  - Tabella di assegnazione delle ore produttive per ciascun membro del team, accumulate in totali per persona e per ruolo;
  - Distribuzione delle ore per ruolo, sotto forma di donut chart;
  - Distribuzione delle ore per la coppia risorsa-ruolo, sotto forma di grafico a barre sovrapposte (così da poter assumere più ruoli per sprint);
  - Preventivo economico dello *sprint*$_G$;

- Tabella riassuntiva con ore e budget spesi e restanti;
- Pie chart con la stima delle ore spese sul totale;
- Pie chart con la stima del budget sul totale;
- Ore rimanenti per la coppia risorsa-ruolo.

La dashboard è stata progettata per affiancare il responsabile nella stesura delle seguenti sezioni del *Piano di Progetto*:

- Preventivo;
- Consuntivo.

Preventivo Il responsabile può duplicare il foglio di calcolo dello *sprint*$_G$ precedente e modificare le seguenti informazioni:

- **Sprint-ID**, dove ID corrisponde al numero dello *sprint*$_G$;
- I riferimenti temporali, nel formato "dal aaaa-mm-gg al aaaa-mm-gg";
- **Preventivo orario:** per ciascuna coppia risorsa-ruolo, il responsabile deve inserire nella cella apposita le ore produttive previste;
- La tabella "preventivo economico" si aggiorna dinamicamente.

Per esportare le tabelle in formato *csv*$_G$, il team ha creato un foglio di calcolo aggiuntivo chiamato "Export". Il responsabile può copiare una tabella e incollarla su questo foglio, tramite le combinazioni di tasti "Ctrl+C" e "Ctrl+Shift+V". Una volta scaricato il file csv, il responsabile può convertire i dati in *LaTeX*$_G$ e inserirli nel *Piano di Progetto*.

I grafici sono generati automaticamente e si dividono in due categorie:

- **Grafici a torta 3D:** possono essere scaricati direttamente in formato PDF;
- **Grafici a barre sovrapposte:** devono essere scaricati in SVG e poi convertiti in PDF.

Il PDF è un formato vettoriale, il che significa che le immagini possono essere scalate senza perdita di qualità. Questo è particolarmente utile per diagrammi e grafici, poiché mantengono la nitidezza anche se ingranditi. Inoltre, il formato PDF è compatibile e facilmente integrabile con *LaTeX*$_G$.

Nella sezione rendicontazione ore, il responsabile deve inserire:

- Le date di inizio e fine dello *sprint*$_G$;
- I ruoli di ciascun componente del team;
- Un link al questionario di valutazione dello *sprint*$_G$.

Il questionario è un modulo *Google Moduli*$_G$ che può essere creato all'interno della dashboard. Il questionario ha la seguente struttura:

- **Titolo:** Valutazione Sprint-ID, dove ID corrisponde al numero dello *sprint*$_G$;
- **Descrizione:** Questionario per la valutazione dello sprint-ID;

- **Domande** (scelta multipla, scala lineare da 1 a 10, risposta breve, scala lineare da 1 a 5, paragrafo):
  - "Come ti è sembrata l'organizzazione dello sprint?";
  - "Come si potrebbe migliorare la pianificazione?";
  - "Sapevi sempre cosa fare nel tuo ruolo?";
  - "Spiega i motivi della risposta precedente (organizzazione, inesperienza, ecc.)";
  - "Il numero di riunioni è stato adeguato?";
  - "Le riunioni sono state organizzate con il giusto preavviso?";
  - "Come ti è sembrata la conduzione dei meeting interni?";
  - "Come ti è sembrata la conduzione dei meeting esterni?";
  - "Quanto ti sei impegnato/a in questo sprint?";
  - "Qual è stato il rapporto ore spese/ore produttive?";
  - "Quali azioni correttive avvieresti dal prossimo sprint?".

Dopo aver creato un nuovo modulo, il responsabile può utilizzare la funzione "Importa domande". Questa feature consente di importare quesiti da un modulo esistente. Cliccando il pulsante "Invia", è possibile inoltre copiare il link da incollare nel foglio di calcolo. Nella dashboard *Google Sheets*$_G$ viene aggiunto in automatico un foglio contenente le risposte al questionario; i fogli degli sprint precedenti possono essere nascosti.

Consuntivo Tutte le tabelle del consuntivo vengono aggiornate automaticamente in base alla rendicontazione delle ore. Di seguito sono riporate le tre tabelle che compongono il consuntivo:

- **Consuntivo orario:** il team ha definito una formula dinamica che somma le ore produttive per la coppia risorsa-ruolo. In questo modo è possibile automatizzare il calcolo del consuntivo anche quando i membri del team assumono più ruoli. La somma delle ore produttive per la coppia risorsa-ruolo è arrotondata per difetto;
- **Ore rimanenti** per la coppia risorsa-ruolo: viene calcolata la differenza tra le ore rimanenti al termine dello *sprint*$_G$ precedente e le ore impiegate nello *sprint*$_G$ attuale;
- **Consuntivo economico**, formato dai seguenti campi:
  - Ruolo;
  - Ore per ruolo;
  - Delta ore preventivo - consuntivo: differenza tra le ore preventivate e quelle effettivamente spese;
  - Costo (in €);

  - Delta costo preventivo - consuntivo: differenza tra il costo preventivato e quello effettivo.
  - Ore e budget spesi negli *sprint*$_G$ precedenti;
  - Ore e budget restanti.

Il processo di esportazione di tabelle e grafici segue le stesse regole del preventivo. Tutte le tabelle e i grafici del consuntivo devono essere inseriti nel *Piano di Progetto*. Una volta completata la stesura del consuntivo nel *Piano di Progetto*, il responsabile deve aggiornare le variabili globali nel foglio "Pdp-global":

- **Ultimo-Sprint:** ID, dove ID è il numero dell'ultimo *sprint*$_G$;
- **Preventivo complessivo** (da modificare qualora sia necessaria una ridistribuzione delle ore per ruolo):
  - Ruolo;
  - Ore per ruolo;
  - Ore individuali;
  - Costo orario (in €);
  - Costo totale (in €);
  - Ore e budget restanti, ricavati dal consuntivo economico dell'ultimo sprint.

Se il preventivo complessivo dovesse mutare, sia la tabella che il grafico corrispondente andrebbero aggiornati nel *Piano di Progetto*.

Rendicontazione ore Ciascun foglio di calcolo dello *sprint*$_G$ include una sezione dedicata alla rendicontazione delle ore. La tabella è organizzata come segue:

- Data;
- Membro del team:
  - Ore produttive;
  - Ruolo;
  - Descrizione delle attività.

2.2 Sviluppo

2.2.1 Descrizione

Il *processo*$_G$ di sviluppo contiene le attività e compiti dello *sviluppatore*$_G$ sotto elencate:

- *Analisi dei requisiti*$_G$;
- *Progettazione*$_G$;
- *Codifica*$_G$ e *testing*$_G$.

2.2.2 Analisi dei Requisiti

2.2.2.1 Descrizione L'Analisi dei Requisiti è eseguita dall'Analista, che redige l'omonimo documento *Analisi dei Requisiti*. Il documento considera i seguenti aspetti:

- TODO

TODO

2.2.3 Progettazione

2.2.3.1 Descrizione L'attività, svolta dal Progettista segue quella di analisi e ha il compito di impostare un'*architettura*$_G$ del software capace di soddisfare i requisiti definiti. Il Progettista sviluppa l'architettura attraverso la creazione di unità e di relazioni tra loro, utilizzando opportunamente dei *design pattern*$_G$ architetturali.

TODO

2.2.4 Codifica e testing

2.2.4.1 Descrizione La codifica segue l'attività di progettazione e viene svolta dal Programmatore. Ha lo scopo di trasformare l'architettura prodotta dal Progettista in codice rispettando le norme definite per ottenere codice mantenibile e di qualità. Il testing è una parte stessa dell'attività di codifica, necessaria ad assicurare la correttezza di ciascuna unità software.

TODO

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

Il processo di documentazione registra l'informazione generata da altri processi o attività. Il processo contiene l'insieme di attività che pianificano, producono, modificano, rilasciano e mantengono i documenti legati al progetto.
Il processo consiste nelle seguenti attività:

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

3.1.1.1 Implementazione del processo Questa attività definisce quali documenti saranno generati durante il progetto, definendo per ciascuno:

- Titolo;
- Scopo;

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

3.1.1.2 Progettazione e sviluppo Questa attività consiste nel progettare e redarre ciascun documento nel rispetto degli standard definiti per formato e contenuto, successivamente controllati dal Verificatore.

3.1.1.3 Rilascio Questa attività comincia con l'approvazione finale del documento da parte del Responsabile in carica, e della Proponente nel caso di verbali ad uso esterno. Prosegue con la pubblicazione del documento nel *repository*$_G$ apposito della documentazione.

3.1.2 Lista documenti

I documenti da produrre e mantenere durante il corso del progetto sono:

- *Piano di Progetto*;
- *Norme di Progetto*;
- *Piano di Qualifica*;
- *Analisi dei Requisiti*;
- *Manuale Utente*;
- *Glossario*;
- *Verbali Interni*;
- *Verbali Esterni*.

3.1.3 Ciclo di vita

Il ciclo di vita di un documento è composto dai seguenti eventi:

1. Vengono definite le caratteristiche di base del documento o di una sua parte come da sezione 3.1.1.1;
2. Il Redattore stila una bozza iniziale. Se è necessario l'input di più persone in maniera sincrona, tale bozza viene prodotto in un ambiente condiviso;
3. Prodotta una bozza di tutto il contenuto necessario, il Redattore produce una versione del documento con la forma e i metodi stabiliti in queste norme;
4. Viene sottoposto a verifica il risultato della redazione. Se il Verificatore propone delle modifiche, vengono attuate ritornando alla fase precedente;
5. In seguito a un esito positivo della verifica, se il risultato è un documento completo e che richiede rilascio, viene sottoposto ad un'approvazione finale del responsabile, bloccante in modo analogo alla verifica.

3.1.4 Ambiente di lavoro

3.1.4.1 *LaTeX* Per lo sviluppo della documentazione del gruppo viene utilizzato un *template* *LaTeX* personalizzato. All'interno del template è definito lo stile della pagina iniziale, delle intestazioni e della formattazione generale. Parte del template permette l'uso di comandi personalizzati per favorire la consistenza di termini specifici spesso utilizzati (es.: nomi di documenti, nomi dei membri), inoltre è gestita sempre attraverso il template l'interazione con i termini per il Glossario.
L'utilizzo del template garantisce:

- Il disaccoppiamento di forma e contenuto della documentazione;
- L'uniformità dello stile della documentazione;
- La responsabilità del Redattore è il solo contenuto;
- La possibilità di creare documenti in maniera modulare, conciliata in modo uniforme.

3.1.4.2 *Docker* La compilazione di file LaTeX può differire in base al compilatore utlizzato, il sistema operativo o altre caratteristiche del sistema locale. Per garantirne l'uniformità, la compilazione dei documenti viene effettuata all'interno di un container Docker costruito a partire da un'immagine comune.

3.1.4.3 Google Docs Per scrivere un documento è spesso necessario lavorare in maniera sincrona, Google Docs permette la condivisione e il lavoro contemporaneo di più persone. I limiti del software tuttavia non permettono di generare un documento finale adeguato, per cui le produzioni tramite questo mezzo sono da considerarsi bozza da cui eseguire la coversione.

3.1.5 Struttura documenti

Ciascun documento è fornito di questi elementi:

- Prima pagina:
  - Logo del gruppo;
  - Titolo;
  - Nome del gruppo;
  - Nome del progetto;
  - Versione attuale;
  - Approvatore;
  - Uso del documento (Interno/Esterno);
  - Destinatari del documento;
  - Logo dell'Università di Padova.
- Registro delle modifiche:

  - Versione del documento in seguito alla modifica;
  - Data della modifica;
  - Redattore della modifica (coincide con il Verificatore nel caso di riga associata alla verifica generale, col responsabile del caso di riga associata al rilascio);
  - Verificatore della modifica (coincide con il Responsabile nel caso di riga associata al rilascio);
  - Descrizione della modifica.
- Indice dei contenuti;

3.1.5.1 Verbali

Ad eccezione dei capitoli dichiarati in precedenza, i verbali presentano una struttura differente rispetto a quella degli altri documenti di progetto. Il contenuto dei verbali, sia interni che esterni, è infatti suddiviso nelle seguenti sezioni:

1. **Informazioni**:
   - Orario di inizio dell'incontro;
   - Orario di fine dell'incontro;
   - Mezzo di pianificazione dell'incontro (Mail, Telegram, riunioni precedenti, ecc.);
   - Tipo di incontro (in presenza/da remoto);
   - Descrizione dell'incontro;
   - **Partecipanti**: sezione che include l'elenco dei partecipanti interni e, in caso di riunione con la *Proponente*$_G$, anche esterni. Per ciascun membro del team, si riporta la durata (in ore e minuti) della partecipazione;
   - **Glossario**: paragrafo finalizzato a stabilire la modalità di formattazione dei termini definiti nel *Glossario* e il numero di occorrenze identificate.
2. **Riunione**: i meeting vengono organizzati con lo scopo di rendicontare il lavoro svolto da ciascun membro del gruppo, chiarire eventuali dubbi, mitigare i rischi, intraprendere azioni correttive e pianificare le attività future. Il capitolo relativo alla riunione è suddiviso in due sezioni:
   - **Ordine del giorno**: scaletta degli argomenti generali affrontati durante la riunione;
   - **Discussione e decisioni**: sezione contenente l'elenco cronologico degli argomenti trattati nel corso del meeting. La discussione di ciascuna tematica non è mai fine a sé stessa, ma mira a prendere decisioni consapevoli e a definire un piano d'azione (vedere tabella Todo / In Progress). Durante le riunioni, si valuta anche il progresso delle attività assegnate negli incontri precedenti. Il team può quindi decidere di considerare un task completato, di prolungare la sua data di scadenza o, se necessario, di suddividere l'attività in sotto-task. Nei verbali esterni, alcune sezioni sono organizzate

secondo lo schema "domanda-risposta", per formalizzare l'interazione tra il team e la *Proponente*$_G$.

3. **Tabella di task Todo / In Progress**: Durante le riunioni, interne ed esterne, il team pianifica le attività a breve e medio-lungo termine. Al fine di ottimizzare la fase di creazione dei *ticket*$_G$ su *Jira Software*$_G$, viene redatta una tabella con le azioni da intraprendere o, in alternativa, i task da portare a termine. Ogni voce è affiancata dal codice univoco del *ticket*$_G$ correlato (se presente) nell'*ITS*$_G$ di *Jira*$_G$. L'ID del *ticket*$_G$ è composto dalla stringa ARGO- seguita da un numero univoco autoincrementante. I campi della tabella sono i seguenti:
    - ID del *ticket*$_G$ *Jira*$_G$ associato all'incarico;
    - Descrizione dell'attività;
    - Nome del componente o dei componenti a cui è assegnato il task;
    - Data di scadenza, in formato AAAA-MM-GG per mantenere la coerenza con il *repository*$_G$ documentale e il registro delle modifiche.
4. **Prossima riunione**: sezione contenente la data ed, eventualmente, l'orario della prossima riunione (se pianificata), con annessa breve descrizione dell'ordine del giorno. Nel caso in cui un meeting sia stato organizzato durante l'incontro precedente (e non tramite chat Telegram), il verbale interno deve includere un link al verbale appropriato come mezzo di pianificazione;
5. **Firma del documento**: spazio per la firma del responsabile. In caso di verbale esterno, l'approvazione finale è a carico della *Proponente*$_G$, che produce in output un documento, in formato PDF, firmato e timbrato.

3.1.6 Stile

Di seguito sono elencate la convenzioni stilistiche adottate dalla documentazione del gruppo.

3.1.6.1 Utilizzo del femminile Quando è necessario fare riferimento tramite ruolo di progetto ad un membro del gruppo con il genere femminile, si utilizzano i seguenti termini:

- **Responsabile** è invariato;
- **Amministratrice** al posto di Amministratore;
- **Analista** è invariato;
- **Progettista** è invariato;
- **Programmatrice** al posto di Programmatore;
- **Redattrice** al posto di Redattore;
- **Verificatrice** al posto di Verificatore.

3.1.6.2 Formattazione testo

- **Termini nel Glossario:** indicati in *corsivo* e con una lettera $_G$ in pedice alla fine della parola. In base a ciascun documento tale formattazione può comparire alla sola prima occorrenza (quando il documento ha lo scopo di essere letto dall'inizio alla fine), o in maniera più frequente (quando il documento può essere letto in maniera più frammentata);
- **Nomi di documento:** scritti in *corsivo* con le iniziali di parola maiuscole eccetto preposizioni (es.: *Piano di Progetto*, non *Piano Di Progetto*). Questo per mantenere la coerenza con le *abbreviazioni*$_G$ (es: AdR, PdP, PdQ, NdP);
- **Way of Working:** indicato con le iniziali di parola maiuscole eccetto preposizioni, per mantenere la coerenza con l'abbreviazione WoW usata anche come comando in *LaTeX*$_G$;
- **Nomi di ruolo:** scritti con la lettera iniziale minuscola; anche vocaboli come team, gruppo e fornitore seguono la medesima regola. L'unica eccezione è rappresentata dai seguenti termini:
  - **Proponente:** declinato al femminile e indicato sempre con la lettera iniziale maiuscola, per garantire la massima formalità possibile;
  - **Cliente e Committente:** scritti con la lettera iniziale maiuscola quando si desidera instaurare un rapporto formale con un attore esterno, altrimenti mantengono l'iniziale minuscola. Per esempio, nella frase "il ruolo di cliente è rivestito dall'azienda Zucchetti S.p.A.", la parola "cliente" non richiede la lettera maiuscola.
- **Data:** indicata in formato YYYY-MM-DD nelle tabelle riassuntive, nei titoli e nei nomi dei file. Il formato esteso (esempio: 20 aprile 2024) si utilizza quando la data rientra in un testo discorsivo.

3.1.7 Strumenti

Gli strumenti impiegati nel processo di documentazione sono:

- **Git:** *Version Control System*$_G$ utilizzato dal gruppo;
- **GitHub:** Piattaforma ospite del repository del gruppo;
- **LaTeX:** *markup language*$_G$ per la scrittura di documenti;
- **Docker:** Software per *containerizzazione*$_G$ utilizzato dal gruppo per uniformare la generazione di documenti;
- **Google Docs:** Strumento per la creazione di documenti condivisi, utilizzato per la collaborazione nella redazione di un documento.

3.2 Gestione della configurazione

3.2.1 Scopo

La seguente sezione viene redatta con lo scopo di formalizzare e automatizzare le procedure applicate dal team, durante tutto il ciclo di vita del software, nell'ambito del processo di "configuration management".

3.2.2 Descrizione

Il processo di gestione della configurazione si occupa di definire e gestire le componenti software utilizzate durante l'intero corso del progetto per mantenere la tracciabilità e gestire il versionamento e i rilasci di prodotti software e documenti. Si tratta di un processo di applicazione di procedure amministrative e tecniche finalizzate a:

- Identificare le componenti software del sistema e stabilire un punto di riferimento da cui misurare i progressi nel tempo;
- Controllare le modifiche e i rilasci degli item;
- Mantenere la tracciabilità delle modifiche;
- Garantire la completezza, coerenza e correttezza degli item.

3.2.3 Issue tracking system (ITS)

Al fine di registrare, gestire e monitorare le attività e sottoattività lungo l'intero ciclo di vita del software, il team impiega l'*Issue Tracking System*$_G$ sviluppato da Atlassian: *Jira*$_G$.

3.2.3.1 Ticket

I *ticket*$_G$ possono essere di quattro tipi:

- **Task**: un'attività o un compito specifico da portare a termine entro la fine di uno *sprint*$_G$ e assegnato a un unico membro del team;
- **Sottotask**: un *ticket*$_G$ subordinato che consente di orientarsi verso una scomposizione più granulare del lavoro. Un'attività, ritenuta troppo onerosa per una singola risorsa, può quindi essere suddivisa in più sottotask, associabili a diversi componenti del gruppo;
- ***Bug***$_G$: un *ticket*$_G$ marcato come *bug*$_G$ segnala la presenza di un'anomalia da risolvere tempestivamente, relativamente al prodotto software, alla documentazione o all'infrastruttura di gestione del progetto;
- ***Story***$_G$: detta anche "User Story", rappresenta una funzionalità del sistema, un requisito espresso dal punto di vista dell'utente.

Il layout di un *ticket*$_G$ è formato dai seguenti campi:

- **Riepilogo**: un titolo che riassume brevemente l'incarico associato al *ticket*$_G$;
- **ID**: un codice univoco autoincrementante generato automaticamente dal sistema secondo la formula ARGO-ID;

- **Descrizione**: una descrizione esaustiva dei risultati attesi al completamento del *ticket*$_G$;
- **Assegnatario**: il componente del gruppo a cui è stata assegnato il compito di risolvere il *ticket*$_G$;
- **Epic**: esprime obiettivi generali o grandi porzioni di lavoro che devono essere frammentati. Ogni *ticket*$_G$ può essere associato a un epic;
- ***Sprint*$_G$**: ciascun *ticket*$_G$ può essere correlato a uno *sprint*$_G$, a sua volta scomposto in più epic;
- **Ticket collegati**: *Jira*$_G$ offre una funzionalità, sia nei campi di contesto che nella timeline di pianificazione, per collegare i *ticket*$_G$ tra loro. Di seguito sono riportate le associazioni predefinite:
  - blocca/è bloccato da (queste sono le due dipendenze più comuni tra i task);
  - clona/è clonato da;
  - duplica/è duplicato da;
  - item correlato a.
- **Sviluppo**: un campo di integrazione tra *Jira*$_G$ e *GitHub*$_G$ che permette di monitorare lo stato di avanzamento dello sviluppo, con annessi link ai *branch*$_G$, *commit*$_G$, *pull request*$_G$, *build*$_G$ e *distribuzioni*$_G$ associati al *ticket*$_G$;
- **Stato**: durante il suo ciclo di vita, un *ticket*$_G$ attraversa tre stati: "To Do", "Doing" e "Done".
- **Versioni di correzione**: le versioni rappresentano punti temporali e traguardi intermedi nello svolgimento del progetto. Ciascun *ticket*$_G$ può essere associato a una determinata versione. L'associazione *ticket*$_G$/versione può essere realizzata direttamente dal *backlog*$_G$ del progetto. Le versioni possono trovarsi in uno dei seguenti tre stati:
  - Non rilasciate;
  - Rilasciate;
  - Archiviate.
- **Commenti**: elenco di commenti da affiancare ai messaggi di *commit*$_G$ per contestualizzare le modifiche e ottimizzare il lavoro di *verifica*$_G$;
- **Aggiungi allegato**: oltre ai commenti, è possibile allegare file di vario formato che non necessitano del controllo di versione;
- **Aggiungi un ticket figlio**: ogni *ticket*$_G$ può avere uno o più *ticket*$_G$ subordinati;
- **Azioni**: *Jira*$_G$ offre la possibilità di creare, gestire e monitorare automazioni, come ad esempio la chiusura di un *ticket*$_G$ una volta effettuato il *merge*$_G$ di una *pull request*$_G$;
- **Rilasci**: elenco delle versioni rilasciate a cui il *ticket*$_G$ è associato.

3.2.4 Automazione chiusura ticket

Su *Jira*$_G$, nelle impostazioni del progetto, è presente una sezione denominata "Automazione", a sua volta suddivisa in quattro sottosezioni:

- **Regole**: elenco delle regole definite dall'amministratore; ciascuna regola richiede un trigger di innesco, ossia un evento che avvia l'esecuzione della procedura definita nel corpo della regola. Una volta stabilito il trigger di attivazione, l'amministratore può scegliere una delle seguenti opzioni:
  - THEN: aggiungi un'azione (es: transizione di un *ticket*$_G$ da uno stato all'altro);
  - IF: aggiungi una condizione (es: verifica se lo stato del *ticket*$_G$ è diverso da "Completato");
  - FOR EACH: applica le azioni e le condizioni ad ogni task (es: esamina tutti i *ticket*$_G$ collegati al *ticket*$_G$ che ha attivato la regola ed esegue per ciascuno di essi una determinata azione);
- **Audit log**: cronologia delle automazioni avviate, con dettagli sullo stato di esecuzione della regola, la data di attivazione e gli elementi associati;
- **Modelli**: set di regole predefinite da importare nel progetto;
- **Utilizzo**: numero di automazioni attivate mensilmente.

Il team ha deciso di introdurre una regola personalizzata per effettuare automaticamente la transizione dello stato dei *ticket*$_G$. Quando viene aperta una *pull request*$_G$ finalizzata alla chiusura di un *ticket*$_G$, il titolo della richiesta deve essere corredato dal codice identificativo del *ticket*$_G$ (ARGO-ID). In alternativa, è possibile menzionare il *ticket*$_G$ nel nome del *branch*$_G$ o nei *commit*$_G$ associati alla *pull request*$_G$. Inoltre, l'assegnatario può lasciare un commento nella forma [ARGO-ID], affinché un bot, denominato jira[bot], possa convertire il commento in un link al *ticket*$_G$ *Jira*$_G$. Una volta effettuato il *merge*$_G$ della *pull request*$_G$ su *GitHub*$_G$, il *ticket*$_G$ collegato passerà in automatico allo stato "Completato".

Utilizzando i modelli predefiniti, il gruppo ha aggiunto altre due regole, rispettivamente per:

- Chiudere automaticamente un *ticket*$_G$ quando tutti i *ticket*$_G$ subordinati (task, story, bug, sottotask) sono completati;
- Chiudere automaticamente un *ticket*$_G$ quando tutti i sottotask sono completati.

3.2.5 Pull request

Come riportato nel registro delle modifiche dei documenti, ogni avanzamento di versione deve essere accompagnato da una fase di verifica. Questo vale anche per lo sviluppo del codice sorgente dell'applicazione ChatSQL, poiché quest'ultimo è sottoposto al versionamento. Per garantire un rilascio controllato delle modifiche, il team ha definito delle Branch Protection Rules. I branch che non accettano push dall'ambiente locale sono i seguenti:

- **main:** ramo di produzione;

- **develop:** ramo di sviluppo, disponibile all'interno del repository ChatSQL.

Lo sviluppo in locale avviene nei cosiddetti feature branch. Quando una funzionalità è pronta per l'ambiente condiviso, l'assegnatario propone le modifiche tramite una pull request. La revisione è di competenza del verificatore, che può decidere se approvare la richiesta, proporre ulteriori modifiche o, in caso di conflitti irriducibili, rifiutare la pull request. Sebbene la finalità principale delle pull request sia la verifica del codice, queste possono essere utilizzate anche come spazi di discussione e risoluzione di bug. Se un membro del team incontra delle difficoltà nello sviluppo di una feature, può quindi inviare una richiesta e menzionare il resto del gruppo.

3.2.5.1 Workflow

Il team adotta due tipi di workflow:

- **Git feature branch:** utilizzato nel repository della documentazione. Questo workflow prevede che tutte le funzionalità siano sviluppate in un ramo dedicato anziché nel main branch. L'obiettivo del team è lavorare in un ambiente di *continuous integration*$_G$; di conseguenza, applicando il "feature branch workflow", il ramo principale non dovrebbe mai contenere codice guasto. I comandi essenziali sono i seguenti:
  - git pull origin main;
  - git checkout -b feature-branch: creazione e passaggio automatico al nuovo branch;
  - git add nome-file: aggiunta del file alla *staging area*$_G$;
  - git commit -m "messaggio";
  - git push -u origin feature-branch: invio del branch al repository remoto.
- **Gitflow:** utilizzato nel repository ChatSQL. Questo workflow prevede l'uso di due rami principali: main e develop. Il main contiene il codice sorgente pronto per il rilascio. Il develop, invece, funge da ramo di integrazione per le funzionalità sviluppate in locale. I comandi essenziali sono i seguenti:
  - git pull origin develop;
  - git checkout -b feature-branch;
  - git add nome-file;
  - git commit -m "messaggio";
  - git push -u origin feature-branch.

La differenza principale rispetto al workflow precedente è che durante lo sviluppo, il ramo predefinito (ovvero il branch di destinazione delle pull request) è il develop. Quando si crea una branch di release, invece, il merge deve essere effettuato sul main. Il branch main, infatti, registra la cronologia ufficiale dei rilasci. Nel "gitflow workflow" è disponibile anche un branch di hotfix, che consente di aggiustare rapidamenti i rilasci di produzione.

3.2.5.2 Apertura pull request

Per aprire una pull request dall'interfaccia web di *GitHub*$_G$, il team deve selezionare i seguenti branch:

- **head ref:** il branch di partenza della pull request;
- **base ref:** il branch di destinazione della pull request.

Una volta generata la pull request, il team deve compilare i seguenti campi:

- **Titolo** della pull request: può contenere anche un riferimento al ticket *Jira*$_G$ associato;
- **Descrizione:** un resoconto delle modifiche proposte. Nella descrizione è opportuno inserire il commento [ARGO-ID], dove ID è il numero univoco del ticket;
- **Reviewers:** uno o più verificatori;
- **Assignees:** uno o più membri del team che propongono le modifiche e aggiornano la pull request;
- **Labels:**
  - Amministratore: attività assegnate agli amministratori;
  - Analista: attività assegnate agli analisti;
  - Progettista: attività assegnate ai progettisti;
  - Programmatore: attività assegnate ai programmatori;
  - Responsabile: attività assegnate ai responsabili;
  - Bug: anomalia software;
  - Documentazione: miglioramenti o integrazioni alla documentazione di progetto;
  - Help wanted: richiesta di supporto o assistenza;
  - Hotfix: soluzione immediata a un problema urgente;
  - Task: implementazione di una nuova funzionalità;
  - Ricerca: attività di ricerca (tecnologie, pattern, modelli, best practice);
  - Sviluppo: attività di sviluppo.
- **Projects:** dopo l'adozione di *Jira*$_G$ come *Issue Tracking System*$_G$, il team ha deciso di modificare la funzione della board di GitHub. Invece di registrare gli issue, la board elenca le pull request, suddivise in:
  - No Status;
  - Todo;
  - In Progress;
  - Done.

Nel campo "Projects", il team deve selezionare la board del progetto Argo e la priorità della pull request, che può essere alta, media o bassa.

Se una pull request è stata creata ma non è ancora pronta per essere unita al ramo base, GitHub mette a disposizione un pulsante per convertirla in una bozza. Una volta ultimata la bozza, è possibile marcare la pull request come "pronta per la revisione".

3.2.5.3 Verifica pull request

La lista delle pull request in attesa di revisione è visibile nella project board di GitHub. Per semplificare il lavoro dei verificatori, le pull request sono ordinate per priorità in ordine decrescente. Il contenuto delle pull request è suddiviso in tre sezioni principali:

- L'area di conversazione: uno spazio di discussione funzionale alla risoluzione collaborativa di problemi;
- La cronologia delle modifiche;
- L'elenco dei file modificati: per ciascuna porzione di codice modificata, GitHub visualizza un confronto con il contenuto precedente alla modifica.

Il verificatore può applicare i seguenti filtri:

- Visualizzazione di uno o più commit specifici;
- Filtraggio dei file per estensione (es.: .tex);
- Selezione della modalità di visualizzazione dei diff (differenze tra i file);
- Filtraggio per il nome del file.

Per ogni file modificato, la procedura di revisione è la seguente:

- Selezione di una riga o di una porzione di codice;
- Cliccando sull'icona blu del commento, viene visualizzata una finestra di dialogo;
- Inserimento di un commento descrittivo, positivo o negativo;
- In alternativa, o in aggiunta, inserimento di una "suggestion". Un suggerimento è una modifica che gli sviluppatori possono integrare direttamente nel codice;
- Pubblicazione del singolo commento o aggiunta del commento alla review.

Il verificatore può lasciare anche un commento generale del file. Una volta completata la review, il verificatore deve selezionare una delle seguenti opzioni, lasciando al contempo un commento riassuntivo opzionale:

- **Comment:** fornisce un feedback generale senza approvare esplicitamente la pull request;
- **Approve:** accetta le modifiche proposte;
- **Request changes:** suggerisce aggiustamenti e azioni correttive.

Quando uno sviluppatore apporta delle modifiche sostanziali a una pull request già verificata, deve richiedere nuovamente la revisione.

3.2.5.4 Chiusura pull request

La chiusura delle pull request che riguardano verbali interni ed esterni, o comunque documenti che richiedono un'approvazione pre-rilascio, spetta al responsabile. Questo perché, dopo la revisione del verificatore, è necessaria un'approvazione generale del documento e una firma. GitHub mette a disposizione del team tre modalità di merging e chiusura delle pull request:

- **Create a merge commit:** i commit della pull request vengono aggiunti al ramo base tramite un merge commit;
- **Squash and merge:** i commit della pull request vengono compressi in un unico commit e aggiunti al ramo base;
- **Rebase and merge:** simile a un fast-forward merge che mantiene la cronologia del progetto lineare.

L'opzione scelta dal team è "Squash and merge", in quanto si tratta di una delle soluzioni più diffuse per mantenere la cronologia del progetto pulita nei *repository*$_G$ pubblici.

3.2.6 Versionamento

Il gruppo mantiene un versionamento per la documentazione nel formato:

$$X.Y.Z$$

X Avanza alla approvazione del Responsabile, corrisponde percui ad ogni rilascio;

Y Avanza ad ogni verifica completa del documento;

Z Avanza ad ogni modifica verificata di un documento.

3.2.7 Repository

Il gruppo utilizza due repository, disponibili su *GitHub*$_G$:

- Repository della documentazione: https://github.com/argo-swe/docs;
- Repository del codice sorgente: https://github.com/argo-swe/chatsql.

Il gruppo impiega inoltre, per l'hosting del sito argo-swe.github.io, un repository aggiuntivo, da non considerare parte del workflow principale in quanto aggiornato e mantenuto solo come "vetrina" del gruppo.

- Repository del sito github.io: https://github.com/argo-swe/argo-swe.github.io.

3.2.7.1 Repository *Docs* Il repository contiene il codice sorgente in LaTeX di tutta la documentazione ufficiale generata durante il progetto, oltre all'ambiente utile alla generazione dei file PDF corrispondenti.

Di seguito è riportata la struttura del repository:

- Il file *README.md* illustra brevemente lo scopo del repository ed elenca i componenti del gruppo;
- Il file *.gitignore* evita il tracciamento di file ausiliari e artefatti di compilazione;
- La directory *Logo* contiene le versioni ufficiali del logo del gruppo, in formato SVG e PNG;
- La directory *sources* include il codice sorgente per la documentazione, suddiviso in due sotto-directory:
  - *model* contiene i file di utilizzo globale all'interno della documentazione;
  - *documents* contiene, in maniera ordinata per fasi di progetto, la documentazione ufficiale.
- La directory *tools* contiene:
  - Strumenti *Docker*$_G$ per adottare un ambiente unico ed evitare problemi di compatibilità tra sistemi operativi;
  - Uno script per compilare automaticamente uno o più documenti.
- La directory *.github/workflows* contiene un file YAML che definisce un flusso di lavoro automatizzato. Il *workflow*$_G$ viene attivato ad ogni merge di una pull request sul ramo base ed esegue i seguenti step:
  - Clonazione del repository sorgente (tutta la cronologia, inclusi branch e tag) all'interno dell'ambiente di esecuzione del job (Ubuntu);
  - Salvataggio e ripristino della cache per evitare di scaricare nuovamente le dipendenze;
  - Nella cache viene memorizzata una chiave che include l'hash dei file *docker-compose.yml* e *Dockerfile*. Se il contenuto di uno di questi file cambia, anche la chiave cambia e la cache viene invalidata;
  - Autenticazione nel registro dei contenitori di GitHub attraverso un token di accesso in scrittura;
  - Avvio del contenitore Docker e caricamento dell'immagine su *GHCR*$_G$ (se avviene un "cache miss");
  - Recupero dell'immagine da *GHCR*$_G$ e avvio del contenitore Docker (se avviene un "cache hit");
  - Questa distinzione tra push e pull di un'immagine, in base alla presenza o meno di una chiave in cache, permette di ridurre significativamente i tempi di esecuzione del *workflow*$_G$;
  - Compilazione dei documenti tramite shell interattiva all'interno del contenitore Docker;
  - Rimozione dei verbali esterni, poiché firmati dalla *Proponente*$_G$, dal build output;
  - Pubblicazione dell'artefatto generato durante il processo di build;

- Clonazione del repository di destinazione (argo-swe.github.io) in una directory temporanea accessibile mediante token;
- Caricamento dei file estratti dall'artefatto nella directory temporanea;
- Commit e push dei documenti in formato PDF nel ramo base del repository di destinazione;
- Arresto dell'esecuzione del contenitore Docker.

Il repository contiene un ramo base (main), in cui vengono inserite le versioni verificate e approvate dei documenti. I documenti vengono redatti all'interno di *feature branch*$_G$, i quali richiedono una fase di verifica prima di essere uniti al ramo base.

3.2.7.2 Repository *ChatSQL* Il repository contiene il codice sorgente dell'applicativo ChatSQL, oltre a un *workflow*$_G$ GitHub Actions per l'analisi statica del codice, l'esecuzione dei test e la generazione automatica dell'artefatto. Il workflow è definito da un file YAML archiviato nella directory *.github/workflows*.

TODO. La struttura di questo repository va completata dopo la configurazione finale dell'ambiente.

- Il file *README.md* illustra brevemente lo scopo del repository ed elenca i componenti del gruppo;
- Il file *.gitignore* evita il tracciamento di file ausiliari e artefatti di compilazione;
- Il file *docker-compose.yml* definisce la struttura dei container docker;
- La directory *frontend* include il codice sorgente per la parte applicativa di frontend sviluppato in VueJS;
- La directory *backend* include il codice sorgente per la parte di backend, all'interno della sottocartella *app* il codice è così suddivido:
  - La directory *engine* contiene la parte funzionale relativa la ricerca semantica;
  - La directory *models* contiene i modelli *DTO*$_G$ esposti dall'interfaccia;
  - La directory *routes* contiene la definizione delle rotte dell'interfaccia esposta dal backend.

4 verifica

TODO

5 Validazione

TODO

6 Processi organizzativi

6.1 Gestione

6.1.1 Descrizione

Il processo di gestione contiene le attività e i task che vengono adottati dal Responsabile per il coordinamento del processo.
Il processo consiste nelle seguenti attività:

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

6.1.1.1 Pianificazione Questa attività comprende tutta la programmazione di assegnazione ruoli e attività, scadenze e previsione del periodo corrente e dei successivi.

6.1.1.2 Esecuzione e controllo Il Responsabile provvede a far eseguire e mantenere il risultato della pianificazione, analizzando e risolvendo i problemi sorti durante l'avanzamento. Problemi e soluzioni saranno documentate.
Il Responsabile inoltre si occupa di comunicare con gli *stakeholder*$_G$.

6.1.1.3 Valutazione e approvazione Il Responsabile assicura la soddisfazione dei requisiti del software o la completezza e correttezza della documentazione durante e alla fine dell'esecuzione dei rispettivi processi.

6.1.2 Ruoli

Questo progetto didattico prevede l'assegnazione dei seguenti ruoli, con una rotazione costante e bilanciata che va considerato nella pianificazione.

6.1.2.1 Responsabile TODO

6.1.2.2 Amministratore TODO

6.1.2.3 Analista TODO

6.1.2.4 Progettista TODO

6.1.2.5 Programmatore TODO

6.1.2.6 Verificatore TODO

6.1.3 Comunicazione

6.1.3.1 Comunicazione interna La comunicazione tra i membri del gruppo è gestita attraverso Telegram e Discord.
Attraverso Telegram il gruppo comunica in modo asincrono e generale, pertanto è opportuno per comunicazioni di interesse di tutto il gruppo e di breve contenuto.
Attraverso Discord il gruppo partecipa a chiamate di gruppo, riunioni o meno, e tramite canali testuali divisi per ruolo è l'organizzazione interna di gruppi ristretti è favorita.
Questi strumenti non devono sovrapporsi tuttavia a mezzi di comunicazione e coordinamento, come ad esempio un *Issue Tracking System*$_G$, in quanto le infromazioni riportate tramite questi strumenti sono più difficilmente tracciabili e riferibili in momenti futuri.

6.1.3.2 Comunicazione esterna La comunicazione esterna è gestita dal Responsabile, attraverso il recapito di posta elettronica del gruppo argo.unipd@gmail.com.

7 Accertamento diqualità

7.1 Scopo

Il processo di Certificazione della qualità mira a garantire che i prodotti software e i vari processi coinvolti nel ciclo di vita del progetto rispettino i requisiti definiti e si attengano ai piani stabiliti. L'obiettivo primario dell'assicurazione della qualità è garantire che tutto il lavoro svolto sia conforme agli standard e alle linee guida predefinite. È essenziale stabilire internamente parametri misurabili per valutare il grado di aderenza alle *best practices*$_G$ dell'ingegneria del software, al fine di condurre un'autovalutazione e un miglioramento continuo del processo

7.2 Processo

Per garantire il raggiungimento e il mantenimento degli standard di qualità prefissati, viene adottato il metodo *PDCA*$_G$, che è un modello di gestione iterativo che aiuta a garantire il miglioramento continuo dei processi e dei prodotti all'interno di un'organizzazione e consente di adattarsi a cambiamenti nel lungo periodo. Il PDCA , noto anche come Ciclo di Deming, si divide in 4 fasi interconnesse:

- Plan (Pianificare): in questa fase, vengono definiti gli obiettivi specifici e misurabili da raggiungere, nonché le strategie e le azioni necessarie per realizzarli. È importante identificare chiaramente le risorse disponibili, i tempi e le modalità di implementazione del piano;
- Do (Fare): una volta pianificato, il piano viene messo in pratica. Questa fase coinvolge l'attuazione delle azioni pianificate, l'allocazione delle risorse e l'esecuzione delle attività secondo le specifiche stabilite nella fase precedente;
- Check (Verificare): qui si valutano i risultati ottenuti confrontandoli con gli obiettivi pianificati e gli standard di qualità prefissati. La verifica comporta la raccolta

e l'analisi dei dati pertinenti per determinare se i processi stanno procedendo come previsto e se si stanno raggiungendo gli obiettivi desiderati;

- Act (Agire): sulla base dei risultati della fase di verifica, vengono identificate eventuali discrepanze o opportunità di miglioramento. In questa fase, si attuano le correzioni necessarie e si apportano le modifiche ai processi per eliminare le inefficienze e migliorare le prestazioni complessive.

7.3 Standard di riferimento

TODO.

7.4 Notazione delle metriche

Le metriche vengono identificate in modo univoco secondo questa notazione : M.Tipo.Codice Dove :

- M: identifica una "metrica";
- Tipo: è un numero da 1 a 4 che indica il tipo di metrica che si sta analizzando secondo 4 categorie:
  - 1: Metriche di prodotto e qualità software;
  - 2: Metriche di processo e progetto;
  - 3: Metriche di gestione dei rischi;
  - 4: Metriche di documentazione.
- Codice: è un numero progressivo all'interno delle 4 macroaree precedenti che identifica in modo univoco la metrica in base al tipo.

7.5 Didascalia

Le metriche descritte in seguito saranno definite secondo questi campi:

- Notazione specifica: in base alle norme sopra descritte;
- Nome: nome della metrica;
- Descrizione: descrizione della metrica;
- Caratteristica di riferimento: una o più caratteristiche definite dagli standard di cui la metrica si occupa;
- Motivo: ragione per cui è stata introdotta la metrica;
- Misurazione: formula o strumenti con cui ricavare un valore quantificabile e tracciabile nel tempo.

7.6 Metriche

TODO.

7.6.1 Tipologie

7.6.2 Metriche di prodotto e di qualità del software:

7.6.2.1 Code coverage

- **Notazione specifica**: M.1.1;
- **Nome**: Code coverage;
- **Descrizione**: Serve per valutare la percentuale di codice sorgente di un'applicazione software che è stata eseguita durante l'esecuzione dei test automatizzati. Indica quindi la quantità di codice che viene testata rispetto alla totalità del codice sorgente. Una copertura topologica del test del 100% di tipo code coverage garantisce di aver eseguito almeno una volta tutte le istruzioni, ma non tutti i rami;
- **Caratteristica di riferimento**: Affidabilità e manutenibilità del software;
- **Motivo**: È stata introdotta per valutare l'efficacia dei test automatizzati nel garantire la correttezza e l'affidabilità del software. Una copertura del codice elevata suggerisce una maggiore confidenza nella stabilità e nella qualità del software;
- **Misurazione**: In Python è stato utilizzato lo strumento Coverage.py che permette di eseguire test del codice sorgente e generare report sulla copertura del codice.

7.6.2.2 Test eseguiti su totali

- **Notazione specifica**: M.1.2;
- **Nome**: Test eseguiti su totali;
- **Descrizione**: Questa metrica indica la percentuale di test effettivamente eseguiti rispetto al numero totale di test pianificati per il testing descritti nel Piano di Qualifica;
- **Caratteristica di riferimento**: Efficacia;
- **Motivo**: È stata introdotta per valutare l'efficacia dell'esecuzione dei test pianificati durante il processo di sviluppo del software. Misura quanto bene il team rispetti il piano di test stabilito e se vengono eseguite tutte le attività di testing pianificate;
- **Misurazione**: La metrica viene calcolata utilizzando la seguente formula:

$$\text{Test eseguiti su totali (\%)} = \frac{N_{te}}{N_{tt}} \times 100$$

  dove:

  - N_{te} rappresenta il numero di test effettivamente eseguiti;
  - N_{tt} rappresenta il numero totale di test pianificati.

7.6.2.3 Test superati

- **Notazione specifica**: M.1.3;
- **Nome**: Test superati;
- **Descrizione**: Questa metrica indica la percentuale di test che sono stati superati con successo rispetto al numero totale di test eseguiti;
- **Caratteristica di riferimento**: Efficacia dei test nel rilevare difetti;
- **Motivo**: È stata introdotta per valutare l'efficacia dei test eseguiti nel rilevare eventuali difetti nel software. Misura quanto bene i test eseguiti stiano identificando e segnalando le eventuali anomalie nel comportamento del software;
- **Misurazione**: La metrica viene calcolata utilizzando la seguente formula:

$$\text{Test superati (\%)} = \frac{N_{ts}}{N_{te}} \times 100$$

  dove:

  - N_{ts} rappresenta il numero totale di test che sono stati superati con successo;
  - N_{te} rappresenta il numero di test effettivamente eseguiti.

7.6.2.4 Fallimento dei test

- **Notazione specifica**: M.1.4;
- **Nome**: Fallimento dei test;
- **Descrizione**: Rappresenta la percentuale di test che non superano con successo i criteri di accettazione o di conformità definiti durante la fase di pianificazione dei test. Misura il grado di non conformità del software rispetto alle aspettative stabilite;
- **Caratteristica di riferimento**: Affidabilità e conformità del software ai requisiti specificati;
- **Motivo**: Il fallimento dei test è stato introdotto per valutare il numero e la percentuale di test che non superano con successo i criteri di accettazione o di conformità definiti durante la pianificazione dei test. Identifica i punti deboli del software e delle attività di testing, consentendo di concentrare gli sforzi di correzione e miglioramento dove sono più necessari;
- **Misurazione**: Viene calcolata utilizzando la seguente formula:

$$\text{Fallimento dei test (\%)} = \frac{N_{fr}}{N_{te}} \times 100$$

  dove:

  - N_{fr}: Numero di fallimenti ricevuti;
  - N_{te}: Numero di test eseguiti.

7.6.2.5 Gestione delle operazioni non permesse

- **Notazione specifica**: M.I.5;
- **Nome**: Gestione delle operazioni non permesse;
- **Descrizione**: Appresenta la percentuale di operazioni non consentite o non gestite correttamente dal software durante l'esecuzione;
- **Caratteristica di riferimento**: Affidabilità e robustezza;
- **Motivo**: La gestione delle operazioni non permesse è stata introdotta per valutare la robustezza e l'affidabilità del software nell'affrontare input imprevisti o non validi. Misura quanto bene il sistema sia in grado di gestire eccezioni e situazioni anomale senza interrompere o compromettere il normale flusso di esecuzione;
- **Misurazione**: Viene calcolata come:

$$\text{Gestione delle operazioni non permesse (\%)} = \frac{N_{np}}{N_t} \times 100$$

  dove:

  - N_{np}: rappresenta il numero di operazioni non permesse o non gestite correttamente;
  - N_t: rappresenta il numero totale di operazioni eseguite durante il test.

7.6.2.6 Numero di parametri per funzione

- **Notazione specifica**: M.I.6;
- **Nome**: Numero di parametri per funzione;
- **Descrizione**: Misura il numero medio di parametri passati alle funzioni nel codice sorgente del software;
- **Caratteristica di riferimento**: Manutenibilità e complessità del software;
- **Motivo**: Il numero di parametri per funzione è stato introdotto per valutare la complessità delle funzioni all'interno del software. Un numero elevato di parametri può indicare una scarsa progettazione modulare e può rendere il codice difficile da comprendere, testare e mantenere. Questa metrica aiuta a identificare funzioni che potrebbero beneficiare di una rifattorizzazione per migliorare la qualità del codice;
- **Misurazione**: La metrica viene calcolata utilizzando la seguente formula:

$$\text{Numero di Parametri per Funzione} = \frac{N_p}{N_f}$$

  dove:

  - N_p: Somma del numero di parametri di tutte le funzioni nel codice;
  - N_f: Numero totale di funzioni nel codice.

7.6.2.7 Core size

- **Notazione specifica**: M.1.7;
- **Nome**: Core size;
- **Descrizione**: I core file sono file altamente interconnessi da una catena di dipendenze cicliche, i quali sono maggiormente propensi ad avere difetti. Core size è la percentuale di file con una o più dipendenze che hanno un alto fan-in (numero di moduli che dipendono da essi) ed un alto fan-out (numero di moduli da cui dipendono). Questi file sono critici per la stabilità e la manutenibilità del sistema poiché la loro complessità e interconnessione li rendono più vulnerabili a difetti e più difficili da modificare senza introdurre errori;
- **Caratteristica di riferimento**: Affidabilità, Manutenibilità;
- **Motivo**: Identificare i file critici nel sistema che sono suscettibili a difetti a causa della loro alta interconnessione e complessità. Monitorare il core size aiuta a prevenire la formazione di "nodi critici" che possono influire negativamente sulla qualità del software;
- **Misurazione**: Il calcolo di questa metrica viene effettuato attraverso la libreria networkx per analizzare le dipendenze.

7.6.2.8 Indice di manutenibilità

- **Notazione specifica**: M.1.8;
- **Nome**: Indice di manutenibilità;
- **Descrizione**: L'indice di manutenibilità è una misura che riflette quanto sia facile mantenere, comprendere, modificare e correggere il codice sorgente;
- **Caratteristica di riferimento**: Manutenibilità;
- **Motivo**: Fornire una misura quantitativa della manutenibilità del software, aiutando a identificare aree del codice che potrebbero richiedere rifattorizzazione o miglioramenti per ridurre i costi di manutenzione e migliorare la qualità del software;
- **Misurazione**: Il calcolo è attuato con lo strumento SonarQube.

7.6.2.9 Linee medie di codice per metodo

- **Notazione specifica**: M.1.9;
- **Nome**: Linee medie di codice per metodo;
- **Descrizione**: Le linee medie di codice per metodo è una misura che indica la lunghezza media, in termini di linee di codice, dei metodi o funzioni all'interno del codice sorgente;
- **Caratteristica di riferimento**: Manutenibilità;

- **Motivo**: Valutare la leggibilità e la manutenibilità del codice. Metodi più corti sono generalmente preferibili perché tendono a essere più semplici da comprendere, testare e mantenere. Identificare metodi eccessivamente lunghi può aiutare a focalizzare gli sforzi di rifattorizzazione;
- **Misurazione**: Il calcolo di questa metrica è effettuato con lo strumento SonarQube.

7.6.2.10 Accuratezza della risposta

- **Notazione specifica**: M.1.10;
- **Nome**: Accuratezza della risposta;
- **Descrizione**: L'accuratezza della risposta è una misura della correttezza e precisione con cui il sistema risponde a una interrogazione in linguaggio naturale. La metrica viene valutata in base a a dei test descritti nel *Dizionario dati*$_G$;
- **Caratteristica di riferimento**: Funzionalità, Affidabilità;
- **Motivo**: Garantire che il sistema fornisca risposte corrette e affidabili è cruciale per la soddisfazione dell'utente e il funzionamento efficiente del software. Misurare l'accuratezza delle risposte aiuta a identificare e correggere errori nel sistema, migliorando la qualità complessiva del software;
- **Misurazione**: L'accuratezza della risposta è misurata in base alla distanza della risposta ottenuta dopo la generazione di un prompt su frasi preselezionate e la risposta attesa scritta dall'attore tecnico nel dizionario dati. Questa percentuale l'abbiamo calcolata come :

  $$\text{Accuratezza della risposta} = \frac{R_c}{R_a} \times 100$$

  Dove:

  - R_c: Risposta corretta;
  - R_a: Risposta attesa.

7.6.2.11 Completezza descrittiva

- **Notazione specifica**: M.1.11;
- **Nome**: Completezza descrittiva;
- **Descrizione**: La completezza descrittiva è una misura della qualità e dell'esaustività dei commenti e della documentazione all'interno del codice sorgente;
- **Caratteristica di riferimento**: Manutenibilità;
- **Motivo**: Garantire che il codice sia ben documentato è cruciale per la manutenibilità e la facilità di comprensione del software. Una documentazione completa aiuta gli sviluppatori a comprendere meglio il funzionamento e la logica del codice, riducendo il tempo necessario per la manutenzione e l'aggiornamento;

- **Misurazione**: La completezza descrittiva è ottenuta tramite questa formula:

$$\text{Completezza descrittiva} = \frac{L_{com}}{L_t} \times 100$$

  Dove:

  - L_{com}: Numero di linee di commento;
  - L_t: Numero totale delle righe del codice.

7.6.2.12 Impatto delle modifiche

- **Notazione specifica**: M.1.12;
- **Nome**: Impatto delle modifiche;
- **Descrizione**: L'impatto delle modifiche è una misura che quantifica l'effetto che una modifica al codice sorgente ha su altre parti del sistema. Questa metrica valuta il numero di moduli o componenti che devono essere alterati o verificati a seguito di una modifica, riflettendo la propagazione delle dipendenze e la complessità del sistema;
- **Caratteristica di riferimento**: Manutenibilità, Affidabilità;
- **Motivo**: Ridurre l'impatto delle modifiche è essenziale per mantenere il sistema facilmente manutenibile e affidabile. Un alto impatto delle modifiche può indicare un sistema altamente interconnesso e fragile, dove piccoli cambiamenti possono causare effetti a catena significativi;
- **Misurazione**: L'impatto delle modifiche è misurato calcolando il numero di moduli o componenti influenzati da una modifica secondo la formula:

$$\text{Impatto delle modifiche} = \frac{M_{mod}}{M_t} \times 100$$

  dove:

  - M_{mod}: Numero di moduli modificati;
  - M_t: Numero totale di moduli.

7.6.2.13 Tempo di risposta

- **Notazione specifica**: M.1.13;
- **Nome**: Tempo di risposta;
- **Descrizione**: Questa metruca serve per misurare l'efficienza del prodotto. Il tempo di risposta indica il tempo medio in cui il sistema genera il *prompt*$_G$ in base al comando immesso dall'utente;
- **Caratteristica di riferimento**: Efficienza-Comportamento;
- **Motivo**: Misurare e migliorare il tempo medio in cui il sistema risponde;

7.6.2.14 Efficienza dell'installazione

- **Notazione specifica**: M.1.14;
- **Nome**: Efficienza dell'installazione;
- **Descrizione**: L'efficienza dell'installazione è una metrica che valuta la rapidità con cui il software può essere installato e configurato il sistema;
- **Caratteristica di riferimento**: Efficienza;
- **Motivo**: Un'installazione efficiente è cruciale per garantire che gli utenti possano iniziare a utilizzare il software rapidamente e senza difficoltà. Ridurre il tempo e lo sforzo necessari per installare il software migliora l'esperienza utente complessiva e riduce la possibilità di errori durante l'installazione;
- **Misurazione**: TODO.

7.6.2.15 Requisiti obbligatori soddisfatti

- **Notazione specifica**: M.1.15;
- **Nome**: Requisiti obbligatori soddisfatti;
- **Descrizione**: Questa metrica indica la percentuale di requisiti obbligatori soddisfatti dal software rispetto al totale dei requisiti obbligatori definiti nel documento di specifica dei requisiti;
- **Caratteristica di riferimento**: Funzionalità, Qualità;
- **Motivo**: Misura il grado di adempimento dei requisiti essenziali per il funzionamento del sistema, fornendo una valutazione della completezza del software rispetto alle specifiche stabilite;
- **Misurazione**: La percentuale di requisiti obbligatori soddisfatti è calcolata utilizzando la seguente formula:

$$\text{Requisiti obbligatori soddisfatti (\%)} = \frac{R_o}{T_o} \times 100$$

  dove:
  - R_o: Requisiti obbligatori soddisfatti;
  - T_o: Totale dei requisiti obbligatori.

7.6.2.16 Requisiti opzionali soddisfatti

- **Notazione specifica**: M.1.16;
- **Nome**: Requisiti opzionali soddisfatti;
- **Descrizione**: Questa metrica indica la percentuale di requisiti opzionali soddisfatti dal software rispetto al totale dei requisiti opzionali definiti nel documento di specifica dei requisiti;
- **Caratteristica di riferimento**: Funzionalità, Qualità;

- **Motivo**: Misura il grado di adempimento dei requisiti opzionali, fornendo una valutazione della completezza del software rispetto alle richieste supplementari specificate;
- **Misurazione**: La percentuale di requisiti opzionali soddisfatti è calcolata utilizzando la seguente formula:

$$\text{Requisiti opzionali soddisfatti (\%)} = \frac{R_o}{T_o} \times 100$$

dove:

  - R_o: Numero di requisiti opzionali soddisfatti;
  - T_o: Numero totale dei requisiti opzionali.

7.6.2.17 Branch coverage

- **Notazione specifica**: M.1.17;
- **Nome**: Branch coverage;
- **Descrizione**: Metrica in grado di misurare in percentuale l'esecuzione completa, durante il test, di tutti i percorsi di decisione presenti nel codice;
- **Caratteristica di riferimento**: Efficienza-Comportamento, Manutenibilità;
- **Motivo**: Questa metrica aumenta la probabilità di individuare bug e comportamenti inattesi, migliorando la qualità del software;
- **Misurazione**: Il branch coverage è calcolato con questa formula:

$$\text{Branch coverage} = \frac{R_e}{R_t} \times 100$$

dove:

  - R_e: Numero di rami eseguiti;
  - R_t: Numero totale di rami.

7.6.3 Metriche di processo:

7.6.3.1 Percentuale di metriche soddisfatte

- **Notazione specifica**: M.2.1;
- **Nome**: Percentuale di metriche soddisfatte;
- **Descrizione**: Questa metrica misura la percentuale di metriche che soddisfano i criteri di accettazione specificati rispetto al totale delle metriche definite;
- **Caratteristica di Riferimento**: Qualità del prodotto;
- **Motivo**: Valutare il grado di conformità del prodotto agli standard e requisiti di qualità;

- **Misurazione**: La percentuale di requisiti opzionali soddisfatti è calcolata utilizzando la seguente formula:

$$\text{Percentuale di metriche soddisfatte (\%)} = \frac{M_s}{M_t} \times 100$$

dove:

  - M_s: Numero di metriche soddisfatte;
  - M_t: Numero totale di metriche.

7.6.3.2 Variazione pianificazione task completati

- **Notazione specifica**: M.2.2;
- **Nome**: Variazione pianificazione task completati;
- **Descrizione**: Questa metrica misura la variazione tra il numero di task pianificati e quelli effettivamente completati entro un periodo di tempo specifico. È un indicatore della capacità del team di rispettare le scadenze e le pianificazioni stabilite;
- **Caratteristica di riferimento**: Gestione del progetto, Adempimento delle scadenze;
- **Motivo**: Valutare l'efficacia della pianificazione del progetto e l'aderenza del team alle tempistiche programmate, identificando eventuali deviazioni che potrebbero richiedere interventi correttivi;
- **Misurazione**: La variazione pianificazione task completati è calcolato utilizzando la seguente formula:

$$\text{Variazione pianificazione task completati} = \frac{T_p - T_c}{T_p} \times 100$$

dove:

  - T_p: Numero di task pianificati;
  - T_c: Numero di task completati.

7.6.3.3 Variazione di costo

- **Notazione specifica**: M.2.3;
- **Nome**: Variazione di costo;
- **Descrizione**: Questa metrica misura la variazione tra il costo pianificato e il costo effettivo di un progetto o di una fase di un progetto. È un indicatore dell'efficacia della gestione dei costi e della capacità del team di mantenere il progetto entro il budget stabilito;
- **Caratteristica di riferimento**: Gestione del progetto, Controllo dei costi;
- **Motivo**: Valutare la precisione delle stime dei costi iniziali e la capacità del progetto di rimanere entro il budget, identificando eventuali sforamenti che potrebbero richiedere interventi correttivi;

- **Misurazione**: La variazione di costo è calcolata utilizzando la seguente formula:

$$\text{Variazione di costo (\%)} = \frac{C_p - C_a}{C_p} \times 100$$

  dove:

  - C_p: Costo pianificato;
  - C_a: Costo attuale (effettivo).

7.6.3.4 Variazione temporale

- **Notazione specifica**: M.2.4;
- **Nome**: Variazione temporale;
- **Descrizione**: Questa metrica misura la variazione tra tempo pianificato e tempo effettivo per completare un progetto o una fase di un progetto;
- **Caratteristica di riferimento**: Gestione del progetto, Adempimento delle scadenze;
- **Motivo**: Valutare la capacità del progetto di rispettare le tempistiche programmate, identificando eventuali ritardi e migliorando la gestione del tempo;
- **Misurazione**: La variazione temporale è calcolata utilizzando la seguente formula:

$$\text{Variazione temporale (\%)} = \frac{T_p - T_e}{T_p} \times 100$$

  dove:

  - T_p: Tempo pianificato;
  - T_e: Tempo effettivo.

7.6.3.5 Velocità di verifica dopo una pull request

- **Notazione specifica**: M.2.5;
- **Nome**: Velocità di verifica dopo una *pull request*$_e$;
- **Descrizione**: Questa metrica misura il tempo medio impiegato per verificare e approvare una *pull request*$_e$ dopo che è stata sottomessa. È un indicatore della rapidità e dell'efficienza del processo di revisione del codice;
- **Caratteristica di riferimento**: Efficienza, Manutenibilità;
- **Motivo**: Valutare la rapidità con cui il team di sviluppo riesce a verificare e integrare le modifiche proposte, migliorando il flusso di lavoro e riducendo i tempi di attesa;
- **Misurazione**: La velocità di verifica dopo una pull request è calcolata utilizzando la seguente formula:

$$\text{Velocità di verifica dopo una pull request} = \frac{1}{N} \sum_{i=1}^{N} D_c i - D_a i$$

dove:

- N: Numero di *pull request*$_G$ verificate;
- D_a: Data apertura *pull request*$_G$;
- D_c: Data chiusura *pull request*$_G$.

7.6.3.6 Frequenza di pull request approvate

- **Notazione specifica**: M.2.6;
- **Nome**: Frequenza di *pull request*$_G$ approvate;
- **Descrizione**: Questa metrica misura la frequenza con cui vengono approvate le *pull request*$_G$ e effettuate le unioni nel ramo principare della repository del progetto al giorno. Indica la frequenza con cui il codice viene integrato nel branch principale dopo aver completato una determinata funzionalità o correzione;
- **Caratteristica di riferimento**: Integrazione continua, Efficienza dello sviluppo;
- **Motivo**: Valutare la frequenza di *pull request*$_G$ unite aiuta a comprendere la rapidità con cui le nuove funzionalità o correzioni vengono integrate nel codice principale del progetto, favorendo un flusso di lavoro continuo e una collaborazione efficace tra i membri del team;
- **Misurazione**: La frequenza di *pull request*$_G$ unite è calcolata utilizzando la seguente formula:

$$\text{Frequenza di pull request unite} = \frac{N}{G}$$

dove:

- N: Numero totale di *pull request*$_G$ unite;
- G: Numero di giorni totali dall'inizio del progetto.

7.6.4 Metriche di gestione dei rischi:

7.6.4.1 Rischi inattesi

- **Notazione specifica**: M.3.1;
- **Nome**: Rischi inattesi;
- **Descrizione**: Questa metrica misura il rapporto tra il numero effettivo di rischi non previsti che emergono durante lo sviluppo del software in ciascuno sprint e il numero massimo previsto di rischi per lo stesso periodo;
- **Caratteristica di riferimento**: Gestione dei rischi;
- **Motivo**: Monitorare i rischi inattesi consente di valutare l'efficacia del processo di identificazione e gestione dei rischi, nonché la capacità del team di rispondere in modo tempestivo e efficace a situazioni impreviste;

- **Misurazione**: Il rapporto tra il numero effettivo di rischi non previsti e il numero massimo previsto di rischi per lo sprint secondo le formula:

$$\text{Rischi inattesi} = \frac{NR_{sprint}}{NR_{max}}$$

dove:

  - NR_{sprint}: Numero effettivo di rischi non previsti durante lo sprint;
  - NR_{max}: Numero massimo previsto di rischi per lo sprint.

7.6.4.2 Rischi non previsti su successi

- **Notazione specifica**: M.3.2;
- **Nome**: Rischi non previsti su successi;
- **Descrizione**: Questa metrica rappresenta il rapporto tra il numero di rischi non previsti che emergono durante lo sviluppo del software e il numero di errori di sistema successi. Gli errori di sistema successi sono quelli che sono stati risolti e corretti con successo durante il processo di sviluppo;
- **Caratteristica di riferimento**: Gestione dei rischi, Affidabilità;
- **Motivo**: Questa metrica fornisce una misura della proporzione di rischi non previsti rispetto agli errori di sistema che sono stati risolti con successo. Aiuta a valutare l'efficacia della gestione dei rischi nel ridurre l'incidenza degli errori nel sistema;
- **Misurazione**: Il rapporto dei rischi è calcolato con questa formula:

$$\text{Rischi non previsti su successi} = \frac{R_n}{E_s}$$

dove:

  - R_n: Numero i rischi non previsti;
  - E_s: Numero di errori di sistema avvenuti.

7.6.4.3 Efficienza delle contromisure

- **Notazione specifica**: M.3.3;
- **Nome**: Efficienza delle contromisure;
- **Descrizione**: Indicatore utile per valutare l'efficacia delle azioni correttive intraprese per mitigare i rischi;
- **Caratteristica di riferimento**: Gestione dei rischi;
- **Motivo**: Questa metrica permette di misurare quanto le contromisure adottate siano state in grado di ridurre o eliminare i rischi identificati e quante si sono rivelate incomplete per valutare se un rischio ha bisogno di modifiche correttive;

- **Misurazione**: Il rapporto dei rischi è calcolato con questa formula:

$$\text{Efficienza delle contromisure} = \frac{1}{N}\sum_{i=1}^{N} Xi$$

dove:

- X: Valore compreso tra [0,1] corrispondente alla valutazione a posteriori dell'efficienza delle mitigazioni nei confronti del rischio;
- N: Numero di rischi mitigati.

7.6.5 Metriche per la documentazione:

7.6.5.1 Indice Gulpease

- **Notazione specifica**: M.4.1;
- **Nome**: Indice Gulpease;
- **Descrizione**: L'Indice Gulpease è una metrica di leggibilità di un testo scritto nella lingua italiana. È basato sulla lunghezza delle parole e sulla lunghezza delle frasi;
- **Caratteristica di riferimento**: Qualità del testo;
- **Motivo**: L'Indice Gulpease fornisce una valutazione della leggibilità di un testo, importante per garantire la comprensibilità e l'accessibilità delle informazioni contenute nel documento;
- **Misurazione**: Per garantire la leggibilità della documentazione si applica questa formula:

$$\text{Indice Gulpease} = 89 + \frac{300 \cdot N_f - 10 \cdot N_l}{N_p}$$

dove:

- N_f: Numero totale di frasi;
- N_l: Numero totale di lettere;
- N_p: Numero totale di parole.

Interpretazione risultato: La formula per il calcolo dell'indice di Gulpease restituisce un punteggio in centesimi che determina il grado di comprensibilità rapportato al livello di istruzione del lettore:

- punteggi intorno a 0: testi a leggibilità più bassa;
- punteggi minori di 40: testi di difficile comprensione per lettori in possesso di un diploma di scuola superiore;
- punteggi minori di 60: testi di difficile comprensione per lettori in possesso di licenza di scuola media;
- punteggi minori di 80: testi di difficile comprensione per lettori in possesso di licenza elementare;
- punteggi intorno a 100: i testi con maggiore livello di comprensione.

7.6.5.2 Vocaboli inseriti nel glossario ad ogni sprint

- **Notazione specifica**: M.4.2;
- **Nome**: Vocaboli inseriti nel glossario ad ogni sprint;
- **Descrizione**: Questa metrica misura il numero di nuove parole che vengono introdotte nel glossario del progetto ad ogni sprint. Il glossario del progetto include tutti i termini tecnici, acronimi e altri concetti specifici utilizzati nella documentazione e nelle comunicazioni del team;
- **Caratteristica di riferimento**: Comunicazione, comprensione del linguaggio tecnico e accessibilità;
- **Motivo**: Monitorare il tasso di crescita del glossario del progetto aiuta a valutare la complessità e l'evoluzione del linguaggio tecnico utilizzato all'interno del team e può evidenziare la necessità di formazione aggiuntiva o di chiarimenti sui concetti;
- **Misurazione**: La metrica misura il numero di nuove parole introdotte nel glossario secondo la formula:

$$\text{Parole inserite nel glossario ad ogni sprint (\%)} = \frac{P_a}{P_t} \times 100$$

  dove:

  - P_a: Numero di vocaboli aggiunti nello sprint;
  - P_t: Numero di vocaboli totali.

7.7 Azioni correttive

TODO

Le metriche elencate verranno utilizzate e valutate all'interno di un modello *SPICE*$_G$ al fine di promuovere un ciclo continuo di valutazione e miglioramento. Alla fine di ogni sprint, dove possibile in modo automatico o manualmente per alcune metriche, verranno valutate le aree di maggior difficoltà e quelle su cui è necessario migliorare. Successivamente, saranno implementate azioni correttive mirate per affrontare eventuali problemi riscontrati.

Ad esempio, nel caso delle metriche del questionario interno, verrà valutato il livello di conformità rispetto ai criteri prestabiliti. In caso di risultati inferiori alle aspettative, saranno adottate azioni correttive specifiche. Come ad esempio, se più di 5 attività vengono lasciate fuori dallo sprint (o una a testa per ogni ruolo), lo sprint non verrà concluso e saranno identificate le cause sottostanti per migliorare il processo di pianificazione e gestione delle attività.

8 Strumenti

- **Discord** (https://discord.com) (Ultimo accesso: 2024-05-18) - 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 virtuali dedicate alle conversazioni vocali.

- **GitHub** (https://github.com/) (Ultimo accesso: 2024-05-18) - 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.
- **Google Calendar** (https://calendar.google.com) (Ultimo accesso: 2024-05-18) - 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.
- **Google Gmail** (https://mail.google.com) (Ultimo accesso: 2024-05-18) - Servizio di posta elettronica sviluppato da Google che offre una piattaforma di email, con funzionalità avanzate di organizzazione.
- **Jira** (https://www.atlassian.com/it/software/jira) (Ultimo accesso: 2024-05-18) - Software utilizzato per l'ITS e gli strumenti di organizzazione del lavoro tra i membri del team.
- **Telegram** (https://www.telegram.com) (Ultimo accesso: 2024-05-18) - 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.
- **Docker** - Piattaforma software utilizzata per sviluppare, distribuire e gestire applicazioni in contenitori, permettendo di creare un ambiente isolato per l'esecuzione di processi.
- **LaTeX** - Linguaggio di markup utilizzato per la preparazione di documenti tecnici e scientifici, noto per la sua capacità di gestire la struttura e la formattazione dei documenti in modo flessibile e professionale.
- **Streamlit** - Framework open-source per la creazione di applicazioni web per il machine learning e la data science, che permette di trasformare script Python in applicazioni web interattive in pochi minuti.
- **Visual Studio Code** - Editor di codice sorgente sviluppato da Microsoft che offre funzionalità avanzate per lo sviluppo e la gestione di progetti software.