Norme di Progetto

Gruppo Argo — ***Progetto ChatSQL***

Informazioni sul documento	
Versione	2.0.0
Approvazione	Sebastiano Lewental
Uso	Interno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo Argo

UNIVERSITÀ DEGLI STUDI DI PADOVA

Registro delle modifiche

Ver.	Data	Redazione	Verifica	Descrizione
2.0.0	2024-09-20	Sebastiano Lewental	Sebastiano Lewental	Approvazione e rilascio del documento
1.1.0	2024-09-19	Riccardo Cavalli	Tommaso Stocco	Revisione generale del documento
1.0.3	2024-09-15	Raul Pianon, Riccardo Cavalli	Marco Cristo	Aggiornamento sezioni *Specifica Tecnica* e *Manuale Utente*
1.0.2	2024-08-28	Marco Cristo	Riccardo Cavalli	Aggiornamento sezione test
1.0.1	2024-08-10	Riccardo Cavalli	Mattia Zecchinato, Tommaso Stocco	Aggiornamento workflow ChatSQL e sezione pull request
1.0.0	2024-07-25	Tommaso Stocco	Tommaso Stocco	Approvazione e rilascio del documento
0.0.24	2024-07-24	Mattia Zecchinato	Riccardo Cavalli	Fix alla formattazione
0.0.23	2024-07-22	Riccardo Cavalli, Tommaso Stocco	Mattia Zecchinato	Aggiunta di processi di miglioramento, audit, revisione congiunta e risoluzione problemi
0.0.22	2024-07-20	Riccardo Cavalli	Tommaso Stocco	Aggiornamento sezione progettazione
0.0.21	2024-07-19	Sebastiano Lewental	Riccardo Cavalli	Stesura convenzioni stile CSS in codifica
0.0.20	2024-07-18	Raul Pianon	Riccardo Cavalli	Aggiornamento processi di supporto
0.0.19	2024-07-18	Riccardo Cavalli	Tommaso Stocco	Aggiornamento sezione strumenti
0.0.18	2024-07-17	Riccardo Cavalli	Tommaso Stocco	Espansione sezioni verifica e validazione

Continua nella prossima pagina

Ver.	Data	Redazione	Verifica	Descrizione
0.0.17	2024-07-16	Riccardo Cavalli	Tommaso Stocco	Stesura sezione release
0.0.16	2024-07-16	Tommaso Stocco	Riccardo Cavalli	Modifica struttura processi primari e di supporto
0.0.15	2024-07-13	Raul Pianon	Marco Cristo, Sebastiano Lewental	Formalizzazione struttura *Norme di Progetto* e lettera di presentazione
0.0.14	2024-07-11	Raul Pianon	Tommaso Stocco, Sebastiano Lewental	Espansione delle sezioni descrittive di *Piano di Progetto* e *Analisi dei Requisiti*
0.0.13	2024-07-10	Riccardo Cavalli	Martina Dall'Amico	Revisione sezione accertamento della qualità
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*$_G$

Continua nella prossima pagina

Ver.	Data	Redazione	Verifica	Descrizione
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
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 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 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.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- Capitolato C9 - ChatSQL:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9.pdf
    (Ultimo accesso: 2024-09-20);
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9p.pdf
    (Ultimo accesso: 2024-09-20).
- Slide PD2 - Corso di Ingegneria del Software - Regolamento del Progetto Didattico:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/PD2.pdf
  (Ultimo accesso: 2024-09-20);
- Standard ISO/IEC 12207:1995:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  (Ultimo accesso: 2024-09-20).

1.3.2 Riferimenti informativi

- Documentazione ufficiale di GitHub:
  https://docs.github.com
  (Ultimo accesso: 2024-09-20);
- Documentazione ufficiale di Git:
  https://www.git-scm.com
  (Ultimo accesso: 2024-09-20);

- I workflow di Git a confronto:
  https://www.atlassian.com/git/tutorials/comparing-workflows
  (Ultimo accesso: 2024-09-20);
- Documentazione ufficiale di Jira:
  https://confluence.atlassian.com/jira
  (Ultimo accesso: 2024-09-20);
- Continuous integration:
  https://it.wikipedia.org/wiki/Integrazione_continua
  (Ultimo accesso: 2024-09-20);
- Continuous delivery:
  https://www.atlassian.com/it/continuous-delivery
  (Ultimo accesso: 2024-09-20);
- 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: 2024-09-20);
- 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: 2024-09-20);
- Slide T4 - Corso di Ingegneria del Software - Gestione di Progetto:
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T4.pdf
  (Ultimo accesso: 2024-09-20);
- Normative e standard:
  https://www.humanwareonline.com/project-management/center/differenze-tra-normative-standard
  (Ultimo accesso: 2024-09-20);
- Esempi di metriche di prodotto:
  http://www.colonese.it/00-Manuali_Pubblicatii/07-ISO-IEC9126_v2.pdf
  (Ultimo accesso: 2024-09-20);
- *Piano di Progetto v2.0.0*;
- *Piano di Qualifica v2.0.0*;
- *Analisi dei Requisiti v2.0.0*;
- *Glossario v2.0.0*;
- *Specifica Tecnica v1.0.0*;
- *Manuale Utente v1.0.0*;
- Verbali interni:
  - 2024-04-03;
  - 2024-04-10;
  - 2024-04-16;
  - 2024-04-20;

  - 2024-04-25;
  - 2024-05-02;
  - 2024-05-07;
  - 2024-05-16;
  - 2024-05-23;
  - 2024-05-28;
  - 2024-06-03;
  - 2024-06-14;
  - 2024-06-22;
  - 2024-07-06;
  - 2024-07-10;
  - 2024-07-18;
  - 2024-07-26;
  - 2024-08-01;
  - 2024-08-08;
  - 2024-08-14;
  - 2024-08-19;
  - 2024-08-27;
  - 2024-09-08;
  - 2024-09-16.
- Verbali esterni:
  - 2024-04-09;
  - 2024-05-06;
  - 2024-05-22;
  - 2024-06-07;
  - 2024-07-09;
  - 2024-09-09.

2 Processi primari

2.1 Fornitura

2.1.1 Descrizione

Il *processo*$_G$ di fornitura consiste nell'insieme di attività e compiti svolti 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. L'obiettivo principale del processo è confrontare le aspettative della Proponente con i risultati del fornitore durante lo svolgimento del progetto, mantenendo dunque una metrica oggettiva tra il preventivo e lo stato corrente.
Il processo consiste nelle seguenti attività:

- Selezione e studio di 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 delle ore, la stima dei costi, l'assunzione degli impegni e l'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 i risultati attesi.

2.1.1.4 Esecuzione e controllo Il fornitore attua 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 3.4 e 3.5. Questo avviene in modo continuo e iterativo.

2.1.2 Rapporti con la Proponente

La Proponente Zucchetti S.p.A. fornisce l'indirizzo di posta elettronica del proprio rappresentante, Gregorio Piccoli, attraverso la quale il fornitore può comunicare in via asincrona o pianificare incontri tramite videochiamata.
I tipi di comunicazione tra Proponente e fornitore sono funzionali a diversi aspetti del progetto:

- Raccolta dei requisiti;
- Raccolta di feedback sui risultati conseguiti;
- Presentazioni dell'avanzamento del prodotto software.

L'approccio adottato dalla Proponente durante il corso del progetto, in quanto *didattico*, è quello di mantenere un ruolo da cliente, affidando al fornitore un grado di libertà ampio da cui consegue una maggiore responsabilità nella realizzazione del prodotto.
Ciascun incontro con la Proponente è accompagnato da un Verbale Esterno che ne riassume i punti chiave.

2.1.3 Documentazione fornita

Di seguito viene descritta la documentazione che il gruppo rende disponibile alla Proponente e ai Committenti.

2.1.3.1 Piano di Progetto Il *Piano di Progetto* è il documento che riguarda la gestione e l'organizzazione del gruppo. Descrive l'analisi e la gestione dei rischi e valuta lo stato di avanzamento del progetto, illustrando la pianificazione, il preventivo e il consuntivo di ciascuno *sprint*$_G$. Il documento è diviso nelle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento e del prodotto con eventuali riferimenti normativi e informativi;
2. **Analisi dei rischi**: quali sono i rischi ai quali il gruppo può andare incontro, l'impatto di tali rischi sulle risorse del progetto, le strategie di rilevamento e di mitigazione. L'analisi dei rischi viene suddivisa in base alla categoria del rischio:
    - Rischi tecnologici;
    - Rischi organizzativi;
    - Rischi di natura personale.
3. **Modello di sviluppo**: descrive il modello di sviluppo adottato dal gruppo, elencandone le modalità di attuazione e i punti di forza;

4. **Stima temporale del progetto**: indica la stima delle date delle revisioni di avanzamento. Viene indicata l'ultima data di aggiornamento della sezione;
5. **Stima dei costi**: fornisce una stima preliminare sui costi del progetto. Viene aggiornata a ogni sprint;
6. **Pianificazione**: definisce gli obiettivi che il gruppo si impegna a raggiungere. Viene disposta una sezione di pianificazione per ogni sprint;
7. **Preventivo**: definisce il preventivo per ciascuno sprint. Il preventivo si compone di:
    - Preventivo orario: tabella di distribuzione delle ore preventivate per ciascun membro del gruppo;
    - Distribuzione ore per la coppia risorsa-ruolo: istogramma che indica le ore preventivate per ciascun ruolo;
    - Distribuzione ore per ruolo: aereogramma che indica la distribuzione in percentuale preventivata per i ruoli;
    - Preventivo economico: tabella che riporta il costo del ruolo per il numero di ore preventivate. Riporta il costo totale, ossia il preventivo economico dello sprint.
8. **Consuntivo**: dispone il consuntivo per lo $sprint_{\sigma}$. Ogni sprint prevede una sezione dedicata al consuntivo. Il consuntivo si compone di:
    - Consuntivo orario: tabella di distribuzione delle ore effettivamente impiegate per ciascun membro del gruppo;
    - Distribuzione ore per la coppia risorsa-ruolo: istogramma che indica le ore impiegate per ciascun ruolo;
    - Distribuzione ore per ruolo: aereogramma che indica la distribuzione in percentuale impiegata per i ruoli;
    - Consuntivo economico: tabella che riporta il costo del ruolo per il numero di ore impiegate. Riporta il totale, ossia il consuntivo economico per lo sprint.
    - Copertura oraria rispetto al totale: aereogramma che indica la percentuale di tempo speso in rapporto al tempo totale per il completamento del progetto;
    - Budget speso rispetto al totale: aereogramma che indica la percentuale di budget speso rispetto al budget totale per il completamento del progetto;
    - Ore rimanenti per la coppia risorsa-ruolo: riporta il numero di ore per ruolo rimanenti ad ogni membro del gruppo. Indica anche le ore rimanenti in totale per membro;
    - Revisione delle attività: indica le attività svolte durante lo sprint;
    - Retrospettiva: indica i risultati del questionario che ogni membro compila al termine di uno sprint per valutarne l'andamento. Oltre a ciò, vengono raccolte delle considerazioni aggiuntive sullo sprint, come ad esempio l'affioramento di rischi;

- Aggiornamento pianificazione e preventivo: definisce le azioni migliorative per il prossimo sprint e una prima pianificazione dei macro obiettivi da esaminare nella pianificazione futura. Delinea inoltre la gestione dei rischi, ossia i rischi rilevati durante lo sprint, il loro impatto e la loro mitigazione.

2.1.3.2 Analisi dei Requisiti L'*Analisi dei Requisiti* è il documento che tratta l'analisi dei requisiti richiesti dal progetto e individuati dal gruppo. Sviluppa inoltre i casi d'uso per questi, ossia le interazioni tra il sistema e l'utente. Il documento è diviso nelle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento e del prodotto con eventuali riferimenti normativi e informativi;
2. **Descrizione**: indica le funzioni principali del prodotto e le caratteristiche dell'utente, ossia i motivi che spingono l'utente ad usare l'applicativo;
3. **Casi d'uso**: indica tutti i casi d'uso individuati dal gruppo durante l'analisi;
4. **Requisiti**: elenco esaustivo dei requisiti del prodotto, indicizzati in base a categorie e fonti da cui proviene il tracciamento.

2.1.3.3 Piano di Qualifica Il *Piano di Qualifica* è il documento che tratta della specifica degli obiettivi di qualità di prodotto e processo. Delinea quindi un insieme di indici di valutazione e validazione del progetto. Il documento è diviso nelle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento e del prodotto con eventuali riferimenti normativi e informativi;
2. **Obiettivi di qualità**: la sezione illustra i valori accettabili e ambiti per le metriche individuate dal team. Viene divisa per metriche e ogni sotto sezione è dotata di una tabella che descrive le metriche con: ID della metrica, nome, valore tollerabile e valore ambito. Le metriche sono così divise:
   - Qualità di processo: indicatori utilizzati per monitorare e valutare la qualità dei processi coinvolti nello sviluppo software;
   - Qualità di prodotto: valutano in modo obiettivo le caratteristiche quantitative e qualitative del software e assicurano la conformità alle aspettative del cliente;
   - Qualità per obiettivo: suddivisione delle metriche in base all'obiettivo (fornitura, sviluppo, gestione dei rischi, ecc.).
3. **Verifica**: sezione dedicata alla verifica e al collaudo del software. Ha lo scopo di rilevare, correggere e prevenire potenziali errori e difetti. È divisa in sezioni in base alla tipologia di analisi. Le sezioni di test sono dotate di tabelle che descrivono i test con: ID, descrizione e stato del test. Gli strumenti di verifica sono suddivisi in:
   - Test di unità: attività di collaudo di singole *unità*$_G$ del software;
   - Test di integrazione: verificano che i diversi moduli, componenti o servizi utilizzati dall'applicazione funzionino in modo integrato;

- Test di sistema: controllano il comportamento del sistema nel suo complesso e verificano che l'applicazione funzioni secondo i requisiti specificati;
- Test di accettazione: sono test formali che precedono il rilascio del prodotto e valutano se l'applicazione è conforme alle aspettative del cliente;
- Checklist: sono strumenti che affiancano il team nell'attività di ispezione. Sono diverse dai test riportati in precedenza poiché la loro tabella è composta solamente da un titolo e una descrizione.

4. **Cruscotto di valutazione della qualità**: sezione dedicata alla dimostrazione del rispetto degli standard di qualità. Ciascuna metrica è accompagnata da una rappresentazione grafica che ne misura l'andamento durante gli *sprint*$_G$. Il grafico in questione rileva, oltre all'andamento generale, tre rette che rappresentano il valore ambito, il valore tollerabile e la tendenza. Ogni grafico è dotato di una descrizione che ne esamina l'andamento. Il cruscotto di qualità viene aggiornato periodicamente per allineare le valutazioni agli *sprint*$_G$ più recenti.

2.1.3.4 Lettera di Presentazione La Lettera di Presentazione è il documento con il quale il gruppo intende candidarsi alla revisione di avanzamento Requirements and Technology Baseline. Vengono indicati i *repository*$_G$ di documentazione e di codice sorgente, assieme all'indirizzo dove è collocato il Proof of Concept. Infine, viene dato l'aggiornamento degli impegni con la stima del preventivo "a finire".

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

2.1.3.6 Specifica Tecnica La *Specifica Tecnica* è il documento che tratta l'architettura del software. Descrive le scelte progettuali e le tecnologie utilizzate per lo sviluppo del prodotto. Il documento è diviso nelle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento con eventuali riferimenti normativi e informativi;
2. **Tecnologie utilizzate**: descrive le tecnologie utilizzate per lo sviluppo del prodotto;
3. **Architettura**: descrive l'architettura del software, specificando i modelli architetturali, la struttura del *repository*$_G$ e il deployment del prodotto;
4. **Progettazione ad alto livello dei componenti**: descrive ad alto livello le funzionalità e i componenti del sistema;
5. **Progettazione di dettaglio front-end**: descrive in dettaglio i componenti del front-end;
6. **Progettazione di dettaglio back-end**: descrive in dettaglio i componenti del back-end. Include anche i diagrammi delle classi e i design pattern utilizzati.

2.1.3.7 Manuale Utente Il *Manuale Utente* è il documento che tratta l'utilizzo del prodotto da parte dell'utente. Descrive le funzionalità del software e come utilizzarle. Il documento è diviso nelle seguenti sezioni:

1. **Introduzione**: indica lo scopo del documento con eventuali riferimenti normativi e informativi;
2. **Requisiti di sistema**: indica i requisiti minimi per l'utilizzo del software;
3. **Installazione**: illustra i passaggi per l'installazione del software;
4. **Avvio**: illustra i passaggi per l'avvio del software;
5. **Istruzioni per l'utilizzo**: descrive le funzionalità del software e come utilizzarle. Le istruzioni sono suddivise in tre sezioni:
    - **Impostazioni generali**: descrive le impostazioni generali del software;
    - **Sezione Utente**: descrive le funzionalità accessibili all'utente non autenticato;
    - **Sezione Tecnico**: descrive le funzionalità accessibili al profilo tecnico.

2.1.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- LaTeX;
- Git;
- Zoom;
- Google Sheets;
- Google Moduli;
- Table Convert Online;
- PDF24 Tools.

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 di progetto* nella stesura delle seguenti sezioni del *Piano di Progetto*:

- Preventivo;
- Consuntivo.

Preventivo Il *responsabile di progetto* 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 di progetto* 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 di progetto* 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 di progetto* 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 di progetto* 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 di progetto* 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 di progetto* 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 seguenti attività e compiti dello *sviluppatore*$_G$:

- Analisi dei Requisiti;
- *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. Il documento di *Analisi dei Requisiti* considera i seguenti aspetti:

- Descrizione del prodotto e caratteristiche ad alto livello;
- Elenco dei casi d'uso;
- Elenco dei requisiti.

2.2.2.2 Casi d'uso La struttura di un caso d'uso è definita come segue:

- UCN - Nome caso d'uso;
- Descrizione;
- Attori principali;
- Precondizioni;
- Postcondizioni;
- Trigger;
- Scenario principale;
- Eventuale scenario alternativo;
- Eventuale inclusioni;
- Eventuali estensioni;
- Eventuali sottocasi d'uso.

Nella scrittura e definizione di un caso d'uso vanno considerati i seguenti punti:

- Le precondizioni devono essere condizioni necessarie per arrivare alla situazione che si presenta nello UC.
- Le precondizioni vengono declinate al passato per identificare la conclusione di un UC, possono includere altri UC;
- Le postcondizioni illustrano cosa succede dopo lo sviluppo dello UC e sono pertanto 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 incontra 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 a uno UC viene riportato in precondizioni, postcondizioni, scenario principale, scenario alternativo, inclusioni ed estensioni;
- I sottocasi d'uso di uno UC vengono inseriti nello stesso file, sotto lo UC principale;
- I sottocasi sono sempre inclusioni del caso d'uso "padre";
- I sottocasi sono indicati con il nome del caso d'uso padre seguito da un punto e dal numero del sottocaso. Ad esempio, un caso d'uso potrebbe essere UC1 e il sottocaso UC1.1;
- Per gli errori si visualizza quasi sempre un messaggio. Pertanto, la postcondizione finale sarà "Viene visualizzato un messaggio con i dettagli dell'errore";
- Il trigger è l'azione che l'utente vuole svolgere e che viene soddisfatta dallo UC.

2.2.2.3 Requisiti I requisiti del prodotto, delineati durante il processo di analisi, si suddividono nelle seguenti categorie:

- **Funzionali (F)**: corrispondono alle funzionalità del sistema;
- **Qualitativi (Q)**: garantiscono la qualità del prodotto;
- **Di vincolo (V)**: indicano le restrizioni e i vincoli normativi del progetto.

Ciasun requisito ha anche un valore di importanza:

- **Obbligatorio (O)**: requisiti inderogabili;
- **Desiderabile (D)**: requisiti non obbligatori, ma comunque di interesse per la *Proponente*$_G$;
- **Opzionale (OP)**: l'implementazione di questi requisiti è lasciata alla discrezione del *fornitore*$_G$.

A partire da questa classificazione, i requisiti vengono identificati da un indice univoco con la seguente struttura:

$$R[Tipologia].[Importanza].[Codice]$$

Dove *R* significa Requisito, *Tipologia* è la sigla associata alla categoria del requisito, *Importanza* è la sigla per il valore di rilevanza e *Codice* è un valore numerico univoco. Descrivere le fonti è una pratica necessaria per orientare sia l'analista che gli altri membri del gruppo verso una definizione più ampia e chiara del requisito; pertanto, ciascuna fonte viene indicata assieme alla descrizione del requisito associato.

2.2.2.4 Grafici I grafici dei casi d'uso sono così composti:

- È presente almeno un attore che interagisce con il sistema. L'attore è indicato con la figura di riferimento in UML ed è dotato di un nome per identificarne il tipo;
- È presente un frame che identifica in quale porzione del sistema si sta svolgendo il caso d'uso. È così composto:
  - All'esterno del frame sono presenti gli attori che interagiscono con il sistema;
  - All'interno vengono rappresentati i casi d'uso e l'interazione che questi hanno tra di loro e con gli attori;
  - Ogni frame è dotato di un'etichetta. Il termine "ChatSQL" si riferisce al sistema nel suo complesso, mentre la notazione "UC[ID]" indica che il diagramma rappresenta i sotto-casi di un caso d'uso specifico.
- Le connessioni tra attore e casi d'uso sono rappresentate da segmenti continui; un attore può interagire con molteplici casi d'uso;
- Le connessioni tra casi d'uso sono rappresentate da segmenti tratteggiati e direzionati;
- Questi segmenti sono dotati di un'etichetta che ne riporta il nome e rispettano lo standard UML per i casi d'uso;
- Gli ellissi che rappresentano i casi d'uso sono dotati di un'etichetta dove viene indicato il caso d'uso e il nome di quest'ultimo;
- Se si realizza un'estensione tra casi d'uso, questa viene esplicitata attraverso un *extension point*$_G$, il quale spiega le condizioni che portano allo scenario alternativo.

2.2.2.5 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Diagrams.net.

2.2.3 Progettazione

2.2.3.1 Scopo

L'attività di progettazione segue quella di analisi e ha lo scopo di definire un'architettura per il prodotto capace di soddisfare i requisiti specificati nell'*Analisi dei Requisiti v2.0.0*. Questa attività viene svolta dai progettisti e prevede l'utilizzo di stili e pattern architetturali. La progettazione ha dunque il compito di assicurare il soddisfacimento delle caratteristiche individuate durante l'analisi. In genere, l'architettura è un sistema formato da componenti, che raggruppano delle unità software composte da uno o più moduli.

Una buona architettura dovrebbe possedere i seguenti attributi di qualità misurabili:

- **Sufficienza**: adeguato livello di soddisfacimento dei requisiti;

- **Modularità**: scomposizione in elementi distinti tra loro e con compiti non sovrapposti;
- **Robustezza**: tolleranza a diverse classi di input;
- **Flessibilità**: capacità di adattarsi a cambiamenti sia adattivi che evolutivi;
- **Riusabilità**: agevola il riuso dei componenti in altri contesti;
- **Efficienza**: utilizzo ottimale delle risorse;
- **Affidabilità**: capacità di mantenere il livello di prestazioni in condizioni specifiche;
- **Semplicità**: minimizza i componenti ed evita l'introduzione di soluzioni o dettagli troppo complessi;
- **Incapsulamento**: nasconde i dettagli implementativi (black-box), con l'obiettivo di ridurre le dipendenze e aumentare la riusabilità e la manutenibilità;
- **Coesione**: gli elementi che concorrono agli stessi obiettivi devono essere raggruppati nello stesso componente;
- **Basso accoppiamento**: limita le dipendenze tra elementi distinti.

2.2.3.2 Proof of Concept

Oltre all'analisi dei requisiti, una precondizione necessaria per la revisione RTB_G è la dimostrazione della validità delle tecnologie scelte dal team. Sulla base delle richieste e delle necessità espresse dalla *Proponente*$_G$, il team valuta i pro e contro delle tecnologie disponibili, considerando fattori come la curva di apprendimento, le prestazioni e la modernità. Una volta ultimata la scelta tecnologica, i progettisti e i programmatori devono realizzare un *Proof of Concept*$_G$. Qualora il team ritenesse le tecnologie selezionate inadeguate, i progettisti e i programmatori saranno responsabili della realizzazione di un ulteriore Proof of Concept (PoC_G), avvalendosi, ove possibile, delle soluzioni già sviluppate. Lo scopo del PoC è:

- Verificare la fattibilità del progetto;
- Dimostrare che il team è in grado di soddisfare i requisiti mediante l'uso delle tecnologie scelte;
- Acquisire un controllo quanto più completo delle tecnologie;
- Ricevere un feedback dalla *Proponente*$_G$ in merito al grado di conformità delle soluzioni sviluppate rispetto al capitolato.

2.2.3.3 Fasi di progettazione

L'attività di progettazione è suddivisa in due fasi:

- **Progettazione logica**;
- **Progettazione di dettaglio**.

2.2.3.4 Progettazione logica

La progettazione logica (o progettazione architetturale) è la prima fase di progettazione e ha lo scopo di delineare l'architettura software. I progettisti definiscono la struttura complessiva del sistema, identificando i componenti principali, le loro responsabilità e le relazioni tra di essi. Per assicurare la coerenza e la consistenza dell'architettura, il team prevede l'uso di stili architetturali, che forniscono una soluzione progettuale di "alto livello". Questa fase è essenziale per garantire che il sistema risponda ai requisiti funzionali e non funzionali. La progettazione logica include le seguenti attività:

- Individuazione dei principali componenti del sistema;
- Definizione delle interfacce e delle modalità di comunicazione tra i componenti;
- Definizione delle strutture dati necessarie per soddisfare i requisiti;
- Stesura iniziale del *Manuale Utente*;
- Progettazione dei test di integrazione;
- Revisione delle soluzioni e degli stili architetturali.

2.2.3.5 Progettazione di dettaglio

La progettazione di dettaglio precede l'attività di codifica e la vincola in modo sostanziale. In questa fase vengono definiti gli attributi e i metodi delle classi, inclusi i relativi tipi di dato. Il team agisce sulle unità architetturali, che devono essere progettate con una complessità tale da consentire la codifica da parte di un singolo *programmatore*. Le unità dell'architettura vengono suddivise in moduli; più unità formano un componente (definito durante la progettazione logica). Lo scopo della progettazione di dettaglio è organizzare e disciplinare la fase di programmazione, documentando le scelte progettuali, assegnando i requisiti a ciascuna unità e definendo gli strumenti necessari per eseguire i test di unità. Il team deve trovare il giusto equilibrio tra scomposizione del sistema e complessità di coordinamento tra i moduli. La progettazione di dettaglio include le seguenti attività:

- Descrizione dettagliata dei componenti, con enfasi sulla definizione specifica di ciascuna unità e modulo;
- Aggiornamento ed espansione del *Manuale Utente*;
- Definizione delle specifiche dei test di integrazione;
- Definizione delle specifiche dei test di unità.

2.2.3.6 Specifica Tecnica

La fase di progettazione deve essere documentata in modo chiaro ed esaustivo, poiché costituisce la base fondamentale per le successive attività di codifica e testing del software. Le scelte progettuali sono dettagliate nel documento di *Specifica Tecnica*, che approfondisce i seguenti temi:

- **Tecnologie utilizzate**: descrizione delle tecnologie, dei framework e delle librerie adottate, con indicazione delle motivazioni che hanno portato alla loro scelta;

- **Architettura generale**: descrizione dell'architettura del sistema e della struttura del *repository*$_G$;
- **Progettazione logica**: fornisce una visione d'insieme delle funzionalità del sistema, definendo i componenti principali senza scendere nei dettagli implementativi;
- **Progettazione di dettaglio**: definizione dei dettagli tecnici e implementativi di ciascun modulo.

Linee guida progettazione dei componenti front-end

- **Descrizione**: breve descrizione delle funzionalità del componente;
- **Sottocomponenti**: elenco dei sottocomponenti e del loro scopo;
- **Tracciamento dei requisiti**: requisiti soddisfatti dal componente.

Linee guida progettazione dei componenti back-end

- **Descrizione**: breve descrizione delle funzionalità del componente;
- **Dettagli della route**:
  - Metodo HTTP (GET, POST, PUT, DELETE);
  - Endpoint;
  - Tags;
  - Modello di risposta;
  - Nome del metodo;
  - Dependency Injection.
- **Esecuzione**: spiegazione ad alto livello del funzionamento del componente e dei possibili esiti della sua esecuzione;
- **Tracciamento dei requisiti**: requisiti soddisfatti dal componente.

Linee guida progettazione di dettaglio front-end

- **Props**: elenco delle proprietà passate al componente;
- **Emits**: elenco degli eventi emessi dal componente;
- **Elementi chiave**: descrizione degli elementi chiave del componente.

Linee guida progettazione di dettaglio back-end

- **Diagramma della classe**: diagramma UML della classe;
- **Descrizione**: breve descrizione della classe;
- **Operazioni** (opzionale): elenco delle operazioni della classe;
- **Attributi** (opzionale): elenco degli attributi della classe;

- **Interfaccia implementata** (opzionale): specifica se la classe implementa un'interfaccia;
- **Estensione** (opzionale): specifica se la classe estende un'altra classe;
- **Metodi** (opzionale): elenco dei metodi della classe;
- **Dipendenze** (opzionale): elenco delle dipendenze della classe.

2.2.3.7 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Diagrams.net.

2.2.4 Codifica

2.2.4.1 Scopo

La codifica, effettuata dai programmatori, ha l'obiettivo di sviluppare il software in conformità con le specifiche architetturali definite dai progettisti. I programmatori traducono la struttura del sistema in codice sorgente utilizzando uno o più linguaggi di programmazione. Questa fase influisce sia sul testing che sulla manutenzione del software. Durante lo sviluppo, è essenziale che i programmatori aderiscano agli standard di programmazione stabiliti.

2.2.4.2 Elementi comuni di stile

Di seguito sono riportati gli elementi comuni di stile:

- **Nomi autoesplicativi** (self-explanatory): i nomi di classi, metodi e variabili devono essere sufficientemente descrittivi da rendere chiaro il loro significato, scopo o contenuto senza necessità di ulteriori commenti esplicativi, salvo nei casi in cui la complessità del codice lo richieda. È fondamentale trovare un equilibrio tra lunghezza e chiarezza descrittiva;
- **Lingua**: utilizzare la lingua inglese per la nomenclatura all'interno del codice sorgente, inclusi i commenti;
- **Abbreviazioni**: utilizzare abbreviazioni e acronimi consolidati o riconosciuti dalla comunità informatica;
- **Sezioni incomplete**: vanno indicate con un commento preceduto dalla dicitura "TODO";
- **Sezioni in attesa di revisione o miglioramento**: vanno indicate con un commento preceduto dalla dicitura "FIXME";
- **Spaziatura**: aggiungere uno spazio attorno agli operatori binari e ternari per migliorare la leggibilità del codice.

2.2.4.3 Stile di codifica: Python

Struttura dei file

- **Import**: insieme di istruzioni per includere moduli, librerie o pacchetti nello script:

  - Librerie standard o di terze parti;
  - Librerie interne o moduli personalizzati.
- **Variabili globali**: variabili accessibili in tutto il programma;
- **Classi**: definiscono un tipo di oggetto o raggruppano funzioni di utilità.

Struttura delle classi

- **Riferimenti**: lista di classi base (ereditarietà) o interfaccia implementata;
- **Costruttore**: metodo speciale chiamato `__init__` (inizializzatore);
- **Metodi**: funzioni definite all'interno di una classe;
- **Attributi**: variabili definite all'interno di una classe.

Documentazione del codice La documentazione del codice sorgente deve essere conforme alle linee guida di Google relative alla scrittura di *docstring*$_G$ in Python (https://google.github.io/styleguide/pyguide.html (Ultimo accesso: 2024-09-20)). In particolare:

- Ogni classe deve essere corredata da:
  - Panoramica del ruolo o dello scopo della classe;
  - Descrizione dettagliata (opzionale);
  - Lista degli argomenti del costruttore;
  - Lista degli attributi, accompagnati da una breve descrizione;
  - Lista dei metodi.
- Ogni metodo deve essere corredato da:
  - Panoramica del comportamento o della funzionalità del metodo;
  - Descrizione dettagliata (opzionale);
  - Lista degli argomenti, accompagnati da una breve descrizione;
  - Lista degli eventuali valori di ritorno, accompagnati da una breve descrizione;
  - Lista delle eventuali eccezioni sollevate, accompagnate da una breve descrizione.

Best practices

- **Import**: importare moduli, librerie o pacchetti all'inizio del file, evitando l'uso dell'asterisco;
- **Self**: utilizzare la parola chiave "self" per riferirsi all'istanza dell'oggetto su cui viene chiamato un metodo;
- **Metodi setter e getter**: utilizzare metodi dedicati per interagire con gli attributi, evitando l'accesso diretto alle proprietà di una classe;

- **Concordanza nomi attributi-argomenti**: gli argomenti di un metodo devono avere lo stesso nome degli attributi corrispondenti nella classe;
- **Lunghezza massima riga (codice)**: 88 caratteri;
- **Lunghezza massima riga (commenti)**: 72 caratteri;
- **Lunghezza massima riga (docstring)**: 72 caratteri.

Convenzioni sintattiche

- **Nomenclatura**: seguire le convenzioni stabilite da PEP 8;
- **Nomi dei file**: *snake_case*$_G$;
- **Nomi delle classi**: *PascalCase*$_G$;
- **Nomi dei metodi**: snake_case;
- **Nomi delle variabili**: snake_case;
- **Nomi delle costanti**: *UPPER_SNAKE_CASE*$_G$;
- **Indentazione**: utilizzare quattro spazi o un carattere di tabulazione;
- **Commenti**: utilizzare il simbolo # per i commenti;
- **Docstring**: utilizzare il simbolo """ per le docstring.

Formattazione del codice Il team aderisce alle convenzioni di formattazione stabilite da PEP 8 (https://peps.python.org/pep-0008/ (Ultimo accesso: 2024-09-20)), la guida ufficiale per lo stile di codifica Python pubblicata dalla Python Software Foundation (PSF). Il rispetto delle linee guida è assicurato e automatizzato tramite uno strumento integrato nell'estensione Python di Visual Studio Code.

2.2.4.4 Struttura front-end

Il front-end contiene le seguenti cartelle principali:

- **public**: risorse statiche come immagini, temi e favicon;
- **src/components**: componenti riutilizzabili;
- **src/components/layout**: componenti che definiscono la struttura generale dell'applicazione (es.: header, footer, sidebar);
- **src/views**: pagine dell'applicazione;
- **src/services**: moduli per la gestione delle interazioni con servizi interni o esterni (chiamate API, autenticazione, messaggi, servizi di utilità);
- **src/composables**: funzioni riutilizzabili;
- **src/assets**: risorse come font e fogli di stile del layout;
- **src/locales**: file JSON per la gestione dell'interfaccia multilingua;
- **src/types**: definizioni dei tipi e delle interfacce;

- **src/router**: definizione delle *route*$_G$.

2.2.4.5 Componenti (Vue.js)

Un SFC (Single File Component) è suddiviso in 3 sezioni:

- **Script**: definisce la logica e il comportamento del componente. Il tag da utilizzare è `<script setup>`, che gestisce automaticamente l'esportazione del componente e migliora la leggibilità e la concisione;
- **Template**: definisce la struttura del componente;
- **Style**: definisce lo stile e l'aspetto del componente.

Queste sezioni appaiono nell'ordine: **script**, **template**, **style**.

Best practices

- **Riusabilità**: creare componenti riutilizzabili, evitando la duplicazione del codice;
- **Numero di componenti per file**: definire un solo componente per ogni file;
- **Funzioni di utilità**: raggruppare le funzioni di utilità in classi dedicate, evitando di includere le "utility" all'interno dei componenti che ne usufruiscono.

Convenzioni sintattiche

- **Nomi dei file**: *PascalCase*$_G$;
- **Indentazione**: utilizzare due spazi.

2.2.4.6 Stile di codifica: JavaScript/TypeScript

Struttura dei file

- **Import**: insieme di istruzioni per includere componenti, servizi o altre dipendenze nello script:
  - Dipendenze esterne;
  - Dipendenze interne.
- **Variabili**;
- **Funzioni**.

Documentazione del codice La documentazione del codice sorgente deve essere conforme al formato *JSDoc*$_G$ (https://jsdoc.app (Ultimo accesso: 2024-09-20)), compatibile sia con JavaScript che con TypeScript.

Convenzioni sintattiche

- **Nomi dei file**: *kebab-case*$_G$;
- **Nomi delle classi**: *PascalCase*$_G$;
- **Nomi delle funzioni**: *camelCase*$_G$;
- **Nomi delle variabili**: camelCase;
- **Nomi delle costanti**: *UPPER_SNAKE_CASE*$_G$;
- **Indentazione**: utilizzare due spazi;
- **Commenti**: utilizzare il simbolo `//` per i commenti;
- **Blocchi JSDoc**: utilizzare il simbolo `/** ... */` per i blocchi JSDoc.

Formattazione del codice Il team aderisce alle convenzioni di formattazione stabilite da *Prettier*$_G$. Prettier può essere integrato in Visual Studio Code ed è compatibile con *ESLint*$_G$.

2.2.4.7 Stile di codifica: CSS

Il team aderisce alle regole di formattazione e stile per CSS definite al seguente indirizzo: https://google.github.io/styleguide/htmlcssguide.html (Ultimo accesso: 2024-09-20). Il codice CSS è organizzato in tre sezioni:

- **Layout**: disposizione generale degli elementi all'interno dell'applicazione; include classi per la gestione di griglie, allineamenti e margini;
- **Componente**: stile di un singolo componente; definisce le sue proprietà distintive e le sue caratteristiche visive;
- **Tema**: personalizzazione dell'aspetto estetico (tema chiaro/scuro).

Convenzioni sintattiche

- **Nomi dei file**: *kebab-case*$_G$;
- **Nomi delle proprietà**: kebab-case;
- **Commenti**: utilizzare il simbolo `/* ... */` per i commenti.

2.2.4.8 Stile di codifica: HTML

Il team aderisce alle regole di formattazione e stile per HTML definite al seguente indirizzo: https://google.github.io/styleguide/htmlcssguide.html (Ultimo accesso: 2024-09-20).

Best practices

- **Attributi ARIA**: utilizzare attributi ARIA per migliorare l'accessibilità.

Convenzioni sintattiche

- **Nomi dei file**: *kebab-case*$_G$;
- **Nomi delle classi**: kebab-case;
- **Nomi degli id**: kebab-case;
- **Commenti**: utilizzare il simbolo `<!-- ... -->` per i commenti.

2.2.4.9 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Git;
- Visual Studio Code.

2.2.5 Testing del codice

Il testing ha lo scopo di verificare il comportamento di ciascuna unità software; viene condotto in parallelo all'attività di codifica. Di seguito sono riportate le tipologie di test che il team si impegna a eseguire come parte integrante del processo di verifica.

2.2.5.1 Tipologie di test

I test eseguiti nella fase antecedente la convalida sono suddivisi in tre categorie principali:

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

2.2.5.2 Test di unità

I test di unità sono una categoria di test del software che si focalizza sulla verifica del corretto funzionamento dei singoli moduli del codice. In genere, una "unità" è la più piccola porzione o segmento (una funzione, un metodo o una classe) testabile in modo autonomo e isolato all'interno del sistema. I test di unità vengono definiti durante la progettazione di dettaglio con l'obiettivo di individuare difetti o malfunzionamenti. La maggior parte dei difetti rilevati tramite analisi dinamica (circa 2/3 del totale) deriva dall'esecuzione dei test di unità. A ogni unità software può essere associata una suite o batteria di test. Nelle fasi iniziali dello sviluppo, il team potrebbe riscontrare delle difficoltà nel testare singole unità, poiché alcuni moduli potrebbero essere incompleti o non disponibili. Pertanto, sono stati introdotti degli strumenti a supporto dell'analisi dinamica:

- **Stub** (o metodo stub): simula una funzionalità del sistema (non ancora codificata e non oggetto di test) come parte del processo di testing di un'unità software;

- **Driver**: componente di supporto che controlla l'esecuzione dei test e aiuta a configurare l'ambiente. Sostituisce il modulo chiamante di una funzionalità sotto test;
- **Mock object**: simula il comportamento di oggetti reali in condizioni controllate; può fornire una risposta preimpostata.

I test di unità si dividono in due categorie:

- **Test funzionali** (black-box): utilizzano dati di ingresso capaci di provocare l'esito atteso. L'obiettivo è accertare che, fornito un determinato input e definita un'aspettativa di output, l'esecuzione di una funzionalità generi il comportamento previsto. I test black-box contribuiscono a misurare quante specifiche e requisiti funzionali sono soddisfatti dal prodotto software. Non possono valutare la correttezza e la completezza della logica interna dell'unità, pertanto devono essere integrati con test strutturali;
- **Test strutturali** (white-box): verificano la struttura interna dell'unità software, esaminando i cammini di esecuzione all'interno dell'unità stessa. L'obiettivo è valutare il funzionamento interno del software, analizzando la logica, i flussi di controllo e la copertura del codice.

Per l'esecuzione dei test funzionali e strutturali, il team utilizza strumenti di automazione appropriati e coerenti con le tecnologie e i linguaggi di programmazione selezionati. Ciascuna unità software deve rispettare le specifiche stabilite durante la progettazione di dettaglio.

2.2.5.3 Test di integrazione

I test di integrazione vengono definiti durante la progettazione architetturale e si basano sui componenti in essa specificati. L'integrazione può essere di tipo incrementale, aumentando il valore funzionale a ogni passo. Questo approccio prevede che i componenti vengano integrati in insiemi già verificati, agevolando la rimozione di eventuali difetti o malfunzionamenti in seguito a cambiamenti. Lo scopo dei test di integrazione è rilevare difetti di design o una scarsa qualità dei test di unità, garantendo che i dati scambiati attraverso ciascuna interfaccia siano conformi alle specifiche. Vi sono due modalità di integrazione incrementale:

- **Bottom-up**: l'integrazione avviene partendo dai moduli più interni al sistema, che sono meno visibili a livello utente e possiedono un minor numero di dipendenze funzionali. Questa strategia richiede l'uso di pochi stub, ma può rallentare la fornitura di servizi di "alto livello". Nell'approccio bottom-up, i driver vengono utilizzati per testare i moduli nei layer inferiori, qualora i moduli nei layer superiori siano ancora incompleti;
- **Top-down**: l'integrazione avviene partendo dai moduli più esterni, che possiedono un maggior numero di dipendenze funzionali e sono quindi di maggior interesse per l'utente. Questa strategia richiede l'uso di molti stub per simulare i moduli mancanti, consentendo di integrare fin da subito le funzionalità di "alto livello".

2.2.5.4 Test di sistema

I test di sistema verificano il comportamento dinamico dell'intero sistema rispetto ai requisiti specificati nel documento di *Analisi dei Requisiti v2.0.0*. Questa fase della verifica ha inizio al completamento dei test di unità e di integrazione, ed è il precursore del collaudo. I test di sistema sono inerentemente funzionali (black-box) e, pertanto, non dovrebbero richiedere conoscenza della logica interna del software. L'ambiente di esecuzione dovrebbe essere simile a quello di produzione, al fine di replicare le condizioni reali di utilizzo del sistema. I test di sistema coprono il software nel suo complesso, verificando tutte le funzionalità integrate e interconnesse.

2.2.5.5 Test: Notazione

I test vengono identificati in modo univoco secondo questa notazione:

$$T.[Tipo].[Codice]$$

dove:

- **T**: indica la parola "test";
- **Tipo**: indica la tipologia di test:
  - **U** (Unità);
  - **I** (Integrazione);
  - **S** (Sistema);
  - **A** (Accettazione).
- **Codice**: è un numero progressivo che identifica in modo univoco i test per ogni tipologia.

2.2.5.6 Test: Stato

Ogni test è corredato da un flag che segnala il suo stato, aggiornato in base alla versione corrente del prodotto software. Il flag può assumere i seguenti valori:

- **N-D**: test non definito (o non disponibile);
- **S**: test eseguito con successo;
- **F**: test fallito.

2.2.5.7 Linee guida test

- Suite di test: i test devono essere raggruppati in base a determinati criteri (es.: componente, modulo, funzionalità);
- Ciascun test all'interno di una suite deve avere responsabilità limitata;
- Setup: ciascun test deve essere indipendente dagli altri e non deve dipendere dall'ordine di esecuzione. Nella fase di preparazione, è necessario inizializzare i dati e le risorse necessarie per l'esecuzione del test, utilizzando dipendenze reali, mock, stub o spy;

- Teardown: ciascun test deve ripristinare lo stato iniziale del sistema;
- Copertura: i test devono coprire tutte le funzionalità del sistema, inclusi i casi limite e le situazioni di errore.

2.2.5.8 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- GitHub;
- Visual Studio Code.

2.2.6 Integrazione software

Il team adotta la pratica di *continuous integration*$_G$ (CI), che consiste nell'allineamento frequente delle modifiche con l'ambiente condiviso. La cronologia dello sviluppo viene mantenuta in un ramo chiamato develop, mentre nel branch principale è registrata la cronologia dei rilasci. A partire dal branch develop, ciascun membro del team può creare un ramo di feature, orientato allo sviluppo di nuove funzionalità, alla definizione di test o alla correzione di bug. Una volta completate le modifiche, viene aperta una pull request per richiedere la verifica del codice. Come definito nella sezione §3.2.5, ciascuna pull request deve essere opportunamente etichettata.

Il processo di integrazione continua implica l'automazione delle fasi di build e test. Pertanto, quando viene aperta una pull request, GitHub Actions avvia un workflow che esegue i test di unità, di integrazione e di sistema definiti nel *Piano di Qualifica v2.0.0*. L'obiettivo del team è garantire che il ramo di sviluppo rimanga stabile e il più possibile privo di difetti.

2.2.6.1 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- GitHub.

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 a 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*;
- *Specifica Tecnica*
- *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 a un'approvazione finale del *responsabile di progetto*, bloccante in modo analogo alla verifica.

3.1.4 Ambiente di lavoro

3.1.4.1 *LaTeX*$_G$ Per lo sviluppo della documentazione del gruppo viene utilizzato un *template*$_G$ *LaTeX*$_G$ 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*$_G$ 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 di progetto* del caso di riga associata al rilascio);
  - Verificatore della modifica (coincide con il *responsabile di progetto* 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 di progetto*. 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 elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Git;
- GitHub;
- LaTeX;
- Docker;
- Google Docs.

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 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 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;
- **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;
- **Assegnatario**: il componente del gruppo a cui è stata assegnato il compito di risolvere il ticket;
- **Epic**: esprime obiettivi generali o grandi porzioni di lavoro che devono essere frammentati. Ogni ticket può essere associato a un epic;
- ***Sprint***$_G$: ciascun ticket può essere correlato a uno sprint, 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 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 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 può essere associato a una determinata versione. L'associazione ticket/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 può avere uno o più ticket 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 è associato.

3.2.4 Automazione chiusura *ticket*$_G$

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 da uno stato all'altro);
  - IF: aggiungi una condizione (es: verifica se lo stato del ticket è diverso da "Completato");
  - FOR EACH: applica le azioni e le condizioni ad ogni task (es: esamina tutti i ticket collegati al ticket 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, il titolo della richiesta deve essere corredato

dal codice identificativo del ticket (ARGO-ID). In alternativa, è possibile menzionare il ticket 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 *Jira*$_G$. Una volta effettuato il *merge*$_G$ della *pull request*$_G$ su *GitHub*$_G$, il ticket collegato passerà in automatico allo stato "Completato".

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

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

In seguito, 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 richiesta o funzionalità;
  - Ricerca: attività di ricerca (tecnologie, pattern, modelli, best practices);
  - Sviluppo: attività di sviluppo;
  - Allineamento: allineamento delle modifiche tra diversi branch in preparazione a un merge nel branch principale;
  - Codifica: attività di codifica (traduzione della progettazione in codice);
  - Test: progettazione e sviluppo di suite di test.
  - Release: modifiche pre-release.
- **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.

Il pulsante di apertura di una pull request prevede due opzioni:

- "Create pull request": apre una pull request pronta per la revisione;
- "Create draft pull request": apre una pull request in draft.

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. Fatta eccezione per i merge commit, tutte le modifiche relative a una pull request dovrebbero essere apportate mentre la pull request è in stato di bozza. In questo modo si previene l'esecuzione eccessiva del workflow GitHub Actions. 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 di progetto*. Questo perché, dopo la revisione del *verificatore*, è necessaria un'approvazione generale del documento e una firma. Per quanto riguarda il repository del codice sorgente, la revisione manuale del verificatore è accompagnata da una verifica automatica mediante GitHub Actions. Entrambe le revisioni sono considerate bloccanti per il merge della pull request. 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 il codice e la documentazione nel seguente formato:

$$X.Y.Z$$

X Avanza alla approvazione del *responsabile di progetto*, corrisponde per cui 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 principale (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 principale.

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*.

Di seguito è riportata la struttura del repository:

- Il file *README.md* illustra i seguenti aspetti:
  - Tecnologie utilizzate;
  - Modalità di avvio del progetto;
  - Formattazione del codice;
  - Variabili d'ambiente;
  - Localizzazione.
- Il file *.gitignore* evita il tracciamento di file ausiliari e artefatti di compilazione;
- Il file *docker-compose.yml* definisce la struttura dei contenitori Docker;
- La directory *frontend* include il codice sorgente per il frontend sviluppato in Vue.js;
- La directory *backend* include il codice sorgente per il backend sviluppato in Python;
- Il workflow GitHub Actions viene attivato nei seguenti casi:
  - Apertura di una pull request (non in draft);
  - Contrassegno di una pull request come pronta per la revisione;
  - Modifiche al ramo associato alla pull request;
  - Merge della pull request (push sul main o sul develop).
- Se la pull request non è in draft, il workflow esegue i seguenti step:
  - Clonazione del repository sorgente 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;
  - Configurazione dell'ambiente Node.js (versione 20);

- Verifica della versione di Node.js installata;
- Installazione delle dipendenze del *frontend*$_G$;
- Linting e formattazione del codice nel *backend*$_G$;
- Type check del codice nel frontend;
- Linting del codice nel frontend;
- Esecuzione dei test di unità nel frontend con Jest;
- Rinominazione della cartella coverage generata da Jest;
- Esecuzione dei test di unità e integrazione nel frontend con Cypress;
- Rinominazione della cartella coverage generata da Cypress;
- Creazione dei file .env.local, necessari per avviare il contenitore Docker;
- Build e caricamento dell'immagine su *GHCR*$_G$ (se avviene un "cache miss");
- Recupero dell'immagine da *GHCR*$_G$ (se avviene un "cache hit");
- Avvio del contenitore Docker (backend);
- Esecuzione dei test di unità e integrazione nel backend;
- Copia del report di copertura dal container Docker al workspace del job;
- Avvio del contenitore Docker (frontend);
- Esecuzione dei test di sistema;
- Caricamento dei report di copertura su CodeCov;
- Arresto dell'esecuzione del contenitore Docker;
- Creazione della directory "ChatSQL/MVP" (se la pull request è stata unita al ramo base);
- Pubblicazione dell'artefatto generato durante il processo di build.

Il repository contiene un ramo principale (main), in cui vengono registrati i rilasci del codice sorgente. Le modifiche vengono apportate all'interno di *feature branch*$_G$, i quali richiedono una fase di verifica prima di essere uniti al ramo di sviluppo (develop).

3.2.8 Release

Come riportato nella sezione dedicata al versionamento, il team adotta una politica di rilascio basata su versioni numeriche, in cui ogni rilascio è identificato da un numero di versione univoco. Attraverso il processo di release management, il gruppo garantisce che il prodotto software e la documentazione siano rilasciati in modo controllato e conforme alle specifiche. Un rilascio può effettuato quando:

- Il branch develop (o il branch di default) ha acquisito funzionalità sufficienti per una release;

- La data di rilascio predeterminata si avvicina.

Il processo di rilascio si compone dei seguenti passaggi:

- Creazione di un branch di release, denominato "release-X.Y.Z";
- Caricamento di eventuali modifiche (correzione di bug o altre attività orientate al rilascio);
- Una volta pronto, il release branch viene unito a quello principale e reintegrato nel ramo di develop (se presente);
- Creazione di un tag per il rilascio, con il numero di versione associato (la formula da seguire è "vX.Y.Z");
- Il passaggio precedente può essere effettuato tramite l'interfaccia web di GitHub o da riga di comando;
- Una volta creato il tag, il team può procedere con la pubblicazione della release;
- Nell'interfaccia web di GitHub, è visibile la sezione "Tags";
- Al suo interno, oltre a visualizzare la lista dei tag, è possibile creare una nuova release (cliccando su "Draft a new release");
- Ciascuna release include le seguenti informazioni:
  - Tag (selezionato dalla lista dei tag esistenti o creato appositamente);
  - Target (branch o commit di riferimento per la release);
  - Note di rilascio (release notes): elencano gli sviluppatori e il registro delle modifiche;
  - Titolo della release;
  - Descrizione della release;
  - Allegati (file binari, documentazione, ecc.);
  - Flag di pre-release (opzionale): la versione viene etichettata come "non pronta per l'ambiente di produzione";
  - Flag di latest-release (opzionale): la versione viene etichettata come "release più recente".
- Pubblicazione della release o salvataggio come bozza.

3.2.9 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Git;
- GitHub;
- Jira;
- Slack.

3.3 Accertamento di qualità

3.3.1 Scopo

Il processo di certificazione della qualità mira a garantire che i prodotti software e i processi coinvolti nel ciclo di vita del progetto siano conformi ai requisiti e alle aspettative. L'obiettivo primario dell'accertamento della qualità è garantire che il lavoro svolto rispetti gli standard e le linee guida. È essenziale stabilire internamente parametri misurabili per valutare il grado di aderenza alle *best practice*$_G$ dell'ingegneria del software, al fine di condurre un controllo e un miglioramento continuo dei processi. L'assicurazione della qualità può avvalersi dei risultati di altri processi di supporto (es.: verifica e validazione).

3.3.2 Garanzia della qualità

Per garantire il raggiungimento e il mantenimento degli standard di qualità prefissati, il team applica il ciclo di *PDCA*$_G$, un metodo di gestione iterativo che contribuisce al controllo e al miglioramento continuo dei processi e dei prodotti all'interno di un'organizzazione; consente inoltre 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 di qualità da conseguire, nonché le strategie e le azioni necessarie per raggiungerli e misurarli. È importante identificare chiaramente le risorse disponibili, i tempi e le modalità di implementazione del piano. La pianificazione aiuta a stabilire obiettivi e processi necessari per fornire i risultati desiderati;
- **Do** (Fare): una volta stabilito, 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 al passaggio precedente. Inoltre, vengono raccolti dati per la generazione di grafici e analisi;
- **Check** (Verificare): in questa fase si valutano i risultati ottenuti confrontandoli con gli obiettivi pianificati e gli standard di qualità prefissati. Attraverso un insieme di indicatori, il team può determinare la qualità dei processi e verificare se i risultati prodotti sono in linea con le attese. I grafici dei dati possono agevolare il processo di test, in quanto è possibile osservare le tendenze di più cicli di PDCA;
- **Act** (Agire): sulla base dei risultati della fase di verifica, vengono identificate eventuali discrepanze, non conformità, opportunità di miglioramento o inefficienze. Durante questa fase, si attuano azioni correttive per migliorare la qualità dei processi e del prodotto.

3.3.3 Notazione delle metriche

Le metriche vengono identificate in modo univoco secondo questa notazione:

$$M.[Tipo].[Codice]$$

dove:

- **M**: indica la parola "metrica";
- **Tipo**: indica il tipo di qualità:
  - **PC**: qualità di processo;
  - **PD**: qualità di prodotto.
- **Codice**: è un numero progressivo che identifica in modo univoco le metriche per ogni tipologia.

3.3.4 Didascalia

Le metriche sono descritte dai seguenti campi:

- **Notazione**: segue le specifiche sopra elencate;
- **Nome**: nome della metrica;
- **Descrizione**: descrizione della metrica;
- **Caratteristiche**: una o più caratteristiche definite dallo standard di riferimento. Questo campo è disponibile solo per le metriche di prodotto;
- **Motivo**: ragione per cui la metrica viene misurata;
- **Misurazione**: formula e/o strumenti con cui calcolare la metrica.

3.3.5 Standard di riferimento per la qualità di prodotto

Per l'identificazione e la classificazione delle metriche, il team segue lo standard ISO/IEC 9126, che suddivide la qualità in: esterna (comportamento del software durante la sua esecuzione), interna (si applica al software non eseguibile) e in uso. Il modello di qualità è suddiviso in sei caratteristiche generali:

- **Funzionalità**: capacità del software di fornire le funzioni necessarie per soddisfare esigenze specifiche operando in determinate condizioni;
- **Affidabilità**: capacità del software di mantenere un determinato livello di prestazioni quando viene usato in condizioni specifiche per un certo periodo di tempo;
- **Usabilità**: capacità del software di essere compreso dall'utente. L'usabilità comprende un insieme di attributi che incidono sullo sforzo necessario per l'uso del prodotto;
- **Efficienza**: capacità del software di fornire prestazioni adeguate in relazione alla quantità di risorse usate;
- **Manutenibilità**: capacità del software di essere modificato per includere correzioni, miglioramenti o adattamenti;
- **Portabilità**: capacità del software di essere trasferito tra ambienti di lavoro diversi, che possono variare sia per hardware che per sistema operativo.

La qualità in uso rappresenta il punto di vista dell'utente sul software. Nel contesto della qualità in uso, le metriche misurano le seguenti caratteristiche:

- **Efficacia**: capacità del software di consentire agli utenti di raggiungere gli obiettivi desiderati con accuratezza e completezza;
- **Produttività**: capacità del software di permettere agli utenti di impiegare una quantità di risorse appropriate in relazione all'efficacia ottenuta in un determinato contesto d'uso;
- **Soddisfazione**: capacità del software di soddisfare gli utenti che ne usufruiscono;
- **Sicurezza**: capacità del software di raggiungere livelli accettabili di rischio, indipendentemente dalla natura del rischio.

3.3.6 Elenco delle metriche

3.3.6.1 Metriche di processo

Percentuale di metriche soddisfatte

- **Notazione specifica**: M.PC.1;
- **Nome**: Percentuale di metriche soddisfatte;
- **Descrizione**: Questa metrica misura la percentuale di metriche che soddisfano i criteri di accettazione rispetto al totale delle metriche. I valori tollerati e ambiti sono specificati nel *Piano di Qualifica v2.0.0*;
- **Motivo**: Valutare il grado di conformità dei processi e del prodotto agli standard di qualità;
- **Misurazione**:

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

  dove:

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

AC (Actual cost)

- **Notazione specifica**: M.PC.2;
- **Nome**: AC (Actual cost);
- **Descrizione**: Questa metrica misura il costo effettivo sostenuto alla data corrente;
- **Motivo**: Controllare i costi e calcolare la spesa effettiva in funzione dell'EAC;
- **Misurazione**: Costo (in €) speso per il progetto.

EV (Earned Value)

- **Notazione specifica**: M.PC.3;
- **Nome**: EV (Earned Value);
- **Descrizione**: Questa metrica misura il valore (in €) delle attività realizzate alla data corrente;
- **Motivo**: Misurare il progresso e calcolare il valore prodotto dal progetto in funzione dell'EAC;
- **Misurazione**:

$$\text{EV} = \textit{BAC} \times (\%\textit{Lavoro completato})$$

  dove:

  - BAC (Budget at Completion): budget previsto per la realizzazione del progetto (riportato nel *Piano di Progetto v2.0.0*).

PV (Planned Value)

- **Notazione specifica**: M.PC.4;
- **Nome**: PV (Planned Value);
- **Descrizione**: Questa metrica misura il costo pianificato (in €) alla data corrente;
- **Motivo**: Controllare i costi e monitorare il progresso;
- **Misurazione**:

$$\text{PV} = \textit{BAC} \times (\%\textit{Lavoro pianificato})$$

  dove:

  - BAC (Budget at Completion): budget previsto per la realizzazione del progetto (riportato nel *Piano di Progetto v2.0.0*).

EAC (Estimated at Completion)

- **Notazione specifica**: M.PC.5;
- **Nome**: EAC (Estimated at Completion);
- **Descrizione**: Questa metrica misura il budget stimato per la realizzazione del progetto (costo sostenuto + stima costo ancora da sostenere);
- **Motivo**: Calcolare il BAC rivisto allo stato corrente del progetto;
- **Misurazione**:

$$\text{EAC} = \textit{AC} + (\textit{BAC} - \textit{EV})$$

  dove:

  - BAC (Budget at Completion): budget previsto per la realizzazione del progetto (riportato nel *Piano di Progetto v2.0.0*).

Variazione del budget tra preventivo e consuntivo

- **Notazione specifica**: M.PC.6;
- **Nome**: Variazione del budget tra preventivo e consuntivo;
- **Descrizione**: Questa metrica misura la variazione tra il costo pianificato e il costo effettivo di un progetto alla data corrente;
- **Motivo**: Valutare con che velocità il team sta spendendo il proprio budget rispetto al preventivo. Identificare eventuali sforamenti che potrebbero richiedere interventi correttivi;
- **Misurazione**:

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

  dove:

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

Variazione del piano tra preventivo e consuntivo

- **Notazione specifica**: M.PC.7;
- **Nome**: Variazione del piano tra preventivo e consuntivo;
- **Descrizione**: Questa metrica misura la variazione tra il numero di task pianificati e quelli completati entro un certo periodo di tempo. La pianificazione dei task è riportata nel *Piano di Progetto v2.0.0*;
- **Motivo**: Valutare la capacità del team di rispettare le scadenze ed evitare ritardi;
- **Misurazione**:

$$\text{Variazione del piano} = \frac{T_p - T_c}{T_p} \times 100$$

  dove:

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

Efficienza temporale

- **Notazione specifica**: M.PC.8;
- **Nome**: Efficienza temporale;
- **Descrizione**: Questa metrica misura il rapporto tra il tempo totale a disposizione (ore di orologio) e il tempo speso in attività produttive (ore produttive);
- **Motivo**: Valutare la capacità del team di utilizzare il tempo in modo efficiente per raggiungere obiettivi e completare attività pianificate. Un'efficienza temporale più alta indica una maggiore produttività e un uso ottimale del tempo a disposizione;

- **Misurazione**:

$$\text{Efficienza temporale (\%)} = \frac{O_r}{O_p} \times 100$$

dove:

  - O_r: Ore di orologio;
  - O_p: Ore produttive.

Frequenza di merge delle pull request

- **Notazione specifica**: M.PC.9;
- **Nome**: Frequenza di merge delle pull request;
- **Descrizione**: Questa metrica misura la frequenza con cui le pull request vengono approvate e unite al ramo base;
- **Motivo**: Valutare la capacità del team di integrare in modo efficiente le nuove funzionalità, modifiche e correzioni nell'ambiente condiviso, favorendo un flusso di lavoro continuo e una collaborazione stretta tra i membri. La chiusura delle pull request include il processo di revisione del codice, l'innesto delle modifiche richieste dal *verificatore* e l'approvazione finale;
- **Misurazione**:

$$\text{Frequenza di merge delle pull request} = \frac{N_{pr}}{T}$$

dove:

  - N_{pr}: Numero totale di pull request approvate e unite al ramo base;
  - T: Periodo di tempo considerato (in giorni).

Indice di stabilità dei requisiti

- **Notazione specifica**: M.PC.10;
- **Nome**: Indice di stabilità dei requisiti;
- **Descrizione**: Questa metrica misura la variazione dei requisiti durante il ciclo di vita del software;
- **Motivo**: Valutare l'impatto delle modifiche ai requisiti e la solidità dell'analisi condotta nel documento di *Analisi dei Requisiti v2.0.0*;
- **Misurazione**:

$$\text{Indice di stabilità dei requisiti} = (1 - \frac{R_a + R_m + R_c}{N_r}) \times 100$$

dove:

  - R_a: Requisiti aggiunti;
  - R_m: Requisiti modificati;
  - R_c: Requisiti cancellati;
  - N_r: Numero totale di requisiti;

Rischi inattesi

- **Notazione specifica**: M.PC.11;
- **Nome**: Rischi inattesi;
- **Descrizione**: Questa metrica misura il numero di rischi non previsti in un determinato periodo;
- **Motivo**: Valutare l'accuratezza della fase di identificazione e analisi dei rischi;
- **Misurazione**:

$$\text{Rischi inattesi} = \textit{N° Rischi inattesi}$$

Efficacia delle contromisure nei rischi

- **Notazione specifica**: M.PC.12;
- **Nome**: Efficacia delle contromisure nei rischi;
- **Descrizione**: Questa metrica misura l'efficacia delle azioni intraprese per mitigare i rischi;
- **Motivo**: Valutare la capacità del team di gestire con successo i rischi emersi;
- **Misurazione**:

$$\text{Efficacia delle contromisure nei rischi} = \frac{R_s}{R_t} \times 100$$

  dove:

  - R_s: Rischi gestiti con successo;
  - R_t: Numero totale di rischi emersi.

3.3.6.2 Metriche di prodotto

Requisiti obbligatori soddisfatti

- **Notazione specifica**: M.PD.1;
- **Nome**: Requisiti obbligatori soddisfatti;
- **Descrizione**: Questa metrica misura la percentuale di requisiti obbligatori soddisfatti rispetto al totale. I requisiti obbligatori sono definiti nel documento di *Analisi dei Requisiti v2.0.0*;
- **Caratteristiche**: Funzionalità;
- **Motivo**: Valutare il grado di adempimento dei requisiti ritenuti essenziali per il funzionamento del sistema;
- **Misurazione**:

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

  dove:

- R_o: Requisiti obbligatori soddisfatti;
- T_o: Numero totale dei requisiti obbligatori.

Requisiti desiderabili soddisfatti

- **Notazione specifica**: M.PD.2;
- **Nome**: Requisiti desiderabili soddisfatti;
- **Descrizione**: Questa metrica misura la percentuale di requisiti desiderabili soddisfatti rispetto al totale. I requisiti desiderabili sono definiti nel documento di *Analisi dei Requisiti v2.0.0*;
- **Caratteristiche**: Funzionalità;
- **Motivo**: Valutare il grado di adempimento dei requisiti desiderabili, fornendo una misura del livello di soddisfazione del cliente;
- **Misurazione**:

$$\text{Requisiti desiderabili soddisfatti (\%)} = \frac{R_d}{T_d} \times 100$$

dove:

- R_d: Requisiti desiderabili soddisfatti;
- T_d: Numero totale dei requisiti desiderabili.

Requisiti opzionali soddisfatti

- **Notazione specifica**: M.PD.3;
- **Nome**: Requisiti opzionali soddisfatti;
- **Descrizione**: Questa metrica misura la percentuale di requisiti opzionali soddisfatti rispetto al totale. I requisiti opzionali sono definiti nel documento di *Analisi dei Requisiti v2.0.0*;
- **Caratteristiche**: Funzionalità;
- **Motivo**: Valutare il grado di adempimento dei requisiti opzionali, fornendo una misura del livello di soddisfazione del cliente;
- **Misurazione**:

$$\text{Requisiti opzionali soddisfatti (\%)} = \frac{R_{op}}{T_{op}} \times 100$$

dove:

- R_{op}: Requisiti opzionali soddisfatti;
- T_{op}: Numero totale dei requisiti opzionali.

Indice Gulpease

- **Notazione specifica**: M.PD.4;
- **Nome**: Indice Gulpease;
- **Descrizione**: Questa metrica misura l'indice di leggibilità di un testo in lingua italiana. I valori sono compresi tra 0 e 100, dove 100 indica una leggibilità più alta mentre 0 una leggibilità più bassa. I punteggi si dividono in:
  - Punteggio inferiore a 80: difficile da leggere per chi possiede la licenza elementare;
  - Punteggio inferiore a 60: difficile da leggere per chi possiede la licenza media;
  - Punteggio inferiore a 40: difficile da leggere per chi possiede un diploma superiore.
- **Caratteristiche**: Usabilità, comprensibilità;
- **Motivo**: Garantire che i documenti di progetto siano comprensbili per la maggior parte dei lettori;
- **Misurazione**:
$$\text{Indice Gulpease} = 89 + \frac{300 \cdot N_f - 10 \cdot N_l}{N_p}$$
dove:
  - N_f: Numero totale di frasi;
  - N_l: Numero totale di lettere;
  - N_p: Numero totale di parole.

Completezza descrittiva

- **Notazione specifica**: M.PD.5;
- **Nome**: Completezza descrittiva;
- **Descrizione**: Questa metrica misura il grado di completezza delle funzionalità descritte nella documentazione del prodotto;
- **Caratteristiche**: Usabilità, comprensibilità;
- **Motivo**: Garantire che gli utenti possano comprendere:
  - Se il prodotto è adeguato alle loro esigenze;
  - Come utilizzare il prodotto per determinati scopi.
- **Misurazione**:
$$\text{Completezza descrittiva} = \frac{F_d}{F_t} \times 100$$
dove:
  - F_d: Numero di funzioni descritte nella documentazione;

- F_t: Numero totale di funzioni previste.

Browser supportati

- **Notazione specifica**: M.PD.6;
- **Nome**: Browser supportati;
- **Descrizione**: Questa metrica misura la percentuale di browser sui quali il prodotto risulta fruibile;
- **Caratteristiche**: Funzionalità, interoperabilità, compatibilità;
- **Motivo**: Valutare la capacità del software di interagire con i browser specificati;
- **Misurazione**:

$$\text{Browser supportati} = \frac{N_{bs}}{N_{bt}} \times 100$$

  dove:
  - N_{bs}: Numero di browser supportati;
  - N_{bt}: Numero totale di browser.

Profondità (click necessari per reperire un'informazione)

- **Notazione specifica**: M.PD.7;
- **Nome**: Profondità (click necessari per reperire un'informazione);
- **Descrizione**: Questa metrica misura il numero di click necessari per raggiungere un obiettivo;
- **Caratteristiche**: Usabilità, accessibilità;
- **Motivo**: Valutare la facilità di navigazione dell'applicazione web;
- **Misurazione**: Calcolare il numero di click necessari per reperire un'informazione seguendo il percorso più ampio.

Ampiezza (opzioni nel menu di navigazione principale)

- **Notazione specifica**: M.PD.8;
- **Nome**: Ampiezza (opzioni nel menu di navigazione principale);
- **Descrizione**: Questa metrica misura il numero di opzioni presenti nel menu di navigazione principale;
- **Caratteristiche**: Usabilità, accessibilità;
- **Motivo**: Valutare la facilità di navigazione dell'applicazione web;
- **Misurazione**: Calcolare il numero di opzioni nel menu di navigazione principale.

Tempo di apprendimento

- **Notazione specifica**: M.PD.9;
- **Nome**: Tempo di apprendimento;
- **Descrizione**: Questa metrica misura il tempo necessario a un utente per comprendere l'utilizzo corretto di una funzione;
- **Caratteristiche**: Usabilità, accessibilità, comprensibilità, apprendibilità;
- **Motivo**: Valutare il design dell'interfaccia e l'esperienza utente;
- **Misurazione**: Test dell'applicazione da parte di un campione eterogeneo di utenti.

Tempo di risposta

- **Notazione specifica**: M.PD.10;
- **Nome**: Tempo di risposta;
- **Descrizione**: Questa metrica misura l'efficienza con cui l'applicazione completa una transazione (task);
- **Caratteristiche**: Efficienza;
- **Motivo**: Misurare e migliorare il tempo medio in cui il sistema risponde a una richiesta dell'utente;
- **Misurazione**: Tempo che intercorre tra l'immissione di un comando e la generazione della risposta da parte del sistema.

Code coverage

- **Notazione specifica**: M.PD.11;
- **Nome**: Code coverage;
- **Descrizione**: Questa metrica misura la copertura dei test, ossia la percentuale di codice sorgente che è stata eseguita durante l'esecuzione dei test automatici.
- **Caratteristiche**: Manutenibilità, testabilità, affidabilità, maturità;
- **Motivo**: Valutare la copertura dei test automatici, al fine di garantire la testabilità del prodotto;
- **Misurazione**:

$$\text{Code coverage} = \frac{L_t}{N_l} \times 100$$

  dove:

  - L_t: Linee di codice testate;
  - N_l: Numero totale di linee di codice.

Adeguatezza delle funzioni sviluppate

- **Notazione specifica**: M.PD.12;
- **Nome**: Adeguatezza delle funzioni sviluppate;
- **Descrizione**: Questa metrica misura il livello di adeguatezza delle funzioni sviluppate rispetto ai requisiti;
- **Caratteristiche**: Funzionalità, adeguatezza;
- **Motivo**: Valutare la conformità ai requisiti funzionali.
- **Misurazione**:

$$\text{Adeguatezza delle funzioni sviluppate} = \frac{F_a}{N_f} \times 100$$

  dove:

  - F_a: Funzioni ritenute adeguate allo svolgimento del task associato;
  - N_f: Numero totale di funzioni sviluppate.

Linee medie di codice per metodo

- **Notazione specifica**: M.PD.13;
- **Nome**: Linee medie di codice per metodo;
- **Descrizione**: Questa metrica misura la lunghezza media, in termini di linee di codice, dei metodi o funzioni all'interno del codice sorgente;
- **Caratteristiche**: Manutenibilità, testabilità, modificabilità, comprensibilità;
- **Motivo**: Verificare che non vi sia mancanza di modularità e chiarezza nel codice. Funzioni e metodi più corti sono generalmente preferibili perché risultano più semplici da comprendere, testare e mantenere;
- **Misurazione**: Il calcolo viene effettuato mediante uno script *Python*$_G$.

Complessità ciclomatica

- **Notazione specifica**: M.PD.14;
- **Nome**: Complessità ciclomatica;
- **Descrizione**: Questa metrica misura il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso;
- **Caratteristiche**: Manutenibilità, comprensibilità;
- **Motivo**: Valutare e limitare la complessità di un programma;
- **Misurazione**:

$$\text{Complessità ciclomatica} = E - N + 2P$$

  dove:

  - E: Numero di archi del grafo;

- N: Numero di nodi del grafo;
- P: Numero di componenti connesse.

Duplicazione del codice

- **Notazione specifica**: M.PD.15;
- **Nome**: Duplicazione del codice;
- **Descrizione**: Questa metrica misura la percentuale di codice duplicato all'interno del software;
- **Caratteristiche**: Manutenibilità;
- **Motivo**: Valutare la progettazione del software. Il codice duplicato è difficile da mantenere e può portare a errori;
- **Misurazione**: Il calcolo viene effettuato con SonarCloud.

Indice di manutenibilità

- **Notazione specifica**: M.PD.16;
- **Nome**: Indice di manutenibilità;
- **Descrizione**: Questa metrica misura la capacità del software di essere modificato, corretto o adattato;
- **Caratteristiche**: Manutenibilità;
- **Motivo**: Fornire una misura quantitativa della manutenibilità del software;
- **Misurazione**: Il calcolo viene effettuato con SonarCloud.

Percentuale di test superati

- **Notazione specifica**: M.PD.17;
- **Nome**: Percentuale di test superati;
- **Descrizione**: Questa metrica misura la percentuale di test eseguiti con successo;
- **Caratteristiche**: Funzionalità, affidabilità, maturità;
- **Motivo**: Garantire la piena copertura dei requisiti funzionali e di qualità;
- **Misurazione**:

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

dove:

- N_{ts}: Numero di test eseguiti con successo;
- N_{tot}: Numero totale di test eseguiti.

Rimozione dei difetti

- **Notazione specifica**: M.PD.18;
- **Nome**: Rimozione dei difetti;
- **Descrizione**: Questa metrica misura la percentuale di difetti identificati e risolti durante lo sviluppo del prodotto;
- **Caratteristiche**: Affidabilità, maturità;
- **Motivo**: Valutare l'efficacia del processo di rilevamento e rimozione dei difetti;
- **Misurazione**:
$$\text{Rimozione dei difetti} = \frac{D_c}{D_t} \times 100$$
dove:
  - D_c: Numero di difetti corretti;
  - D_t: Numero totale di difetti rilevati.

Tolleranza agli errori

- **Notazione specifica**: M.PD.19;
- **Nome**: Tolleranza agli errori;
- **Descrizione**: Questa metrica misura la percentuale di errori che il prodotto è in grado di gestire;
- **Caratteristiche**: Affidabilità, operabilità;
- **Motivo**: Verificare che il software sia in grado di rilevare condizioni di errore e segnalarle con un opportuno messaggio;
- **Misurazione**:
$$\text{Tolleranza agli errori} = \frac{Err_s}{Err_p} \times 100$$
dove:
  - Err_s: Numero di errori gestiti con successo;
  - Err_p: Numero totale di errori previsti.

Impatto delle modifiche

- **Notazione specifica**: M.PD.20;
- **Nome**: Impatto delle modifiche;
- **Descrizione**: Questa metrica misura l'impatto sulla corretta esecuzione del software procurato dalle modifiche al codice;
- **Caratteristiche**: Manutenibilità, stabilità;
- **Motivo**: Valutare la stabilità del prodotto a seguito di cambiamenti. Un alto impatto negativo può indicare un sistema vulnerabile, in cui le modifiche causano effetti a catena significativi;

- **Misurazione**:

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

  dove:

  - I_m: Numero di modifiche che hanno influito negativamente sul corretto funzionamento del software o sulle sue prestazioni;
  - M_t: Numero totale di modifiche eseguite.

3.4 Verifica

3.4.1 Scopo

Il processo di verifica ha lo scopo di determinare se i risultati di un'attività soddisfano i requisiti, le condizioni e i vincoli stabiliti nel *Piano di Qualifica v2.0.0*. Questo processo può includere:

- **Analisi**;
- **Revisione**;
- **Testing**.

I task coinvolti nel processo di verifica sono finalizzati a garantire l'adeguatezza, completezza e coerenza del prodotto. Questi task comprendono:

- Verifica dei processi;
- Verifica dei requisiti software;
- Verifica del design;
- Verifica del codice sorgente;
- Verifica dell'integrazione;
- Verifica della documentazione.

3.4.2 Descrizione

Per assicurare la conformità dei risultati prodotti, a ogni azione di modifica è associato un passo di verifica. L'avanzamento di versione avviene soltanto a valle di verifica e conseguente approvazione. Il processo di revisione viene svolto dai membri impiegati nel ruolo di *verificatore*. Come indicato nella sezione §3.2.5, il gruppo ha definito delle Branch Protection Rules, al fine di garantire un'integrazione controllata delle modifiche all'interno del *repository*$_G$. In linea con le specifiche di GitHub, la verifica non può essere effettuata dallo stesso componente a cui è stato assegnato il task.

3.4.3 Analisi statica

L'analisi statica è un approccio alla verifica che prevede una disamina del software e dei documenti alla ricerca di difetti, senza richiedere l'esecuzione del codice. Può

essere vista come un'attività complementare all'analisi dinamica. Il team utilizza due tecniche di analisi statica:

- **Walkthrough**;
- **Inspection**.

3.4.3.1 Walkthrough

Il walkthrough è una tecnica informale di analisi statica che prevede una lettura critica e approfondita del documento o del codice sorgente. Questo processo coinvolge il *verificatore* e l'autore (un *programmatore* o un *redattore*). Il walkthrough è un'attività collaborativa, che può coinvolgere anche un team di tre o cinque persone. In caso di verifica del codice sorgente, i verificatori lo percorrono simulando possibili esecuzioni. Il walkthrough si articola nelle seguenti fasi:

- **Pianificazione**: il *verificatore* e l'autore si accordano su come procedere con il walkthrough;
- **Lettura**: il codice o il documento viene letto dall'autore, mentre il *verificatore* annota i difetti riscontrati;
- **Discussione**: al termine della lettura, il *verificatore* comunica i problemi rilevati e propone eventuali suggerimenti, con l'obiettivo di correggere i difetti.

3.4.3.2 Inspection

L'inspection (o ispezione) è una tecnica formale di analisi statica che prevede una revisione sistematica e mirata del prodotto. A differenza del walkthrough, l'ispezione utilizza liste di controllo (checklist) per rilevare i difetti. In questo modo è possibile ricercare errori frequenti di programmazione o di altra natura, senza dover analizzare l'oggetto in esame nella sua interezza; pertanto, l'ispezione si concentra sulla verifica dei punti ritenuti critici. Le checklist possono derivare da conoscenze pregresse ottenute tramite walkthrough. L'ispezione si articola nelle seguenti fasi:

- **Pianificazione**: il *verificatore* e l'autore si accordano su come procedere con l'ispezione;
- **Definizione delle checklist**: vengono definiti i punti critici del codice o della documentazione; le liste di controllo vengono aggiornate sulla base dell'esperienza acquisita e degli errori più ricorrenti;
- **Lettura**: il prodotto viene esaminato seguendo le liste di controllo;
- **Correzione dei difetti**: A seguito dell'ispezione, l'autore implementa le modifiche necessarie e intraprende le azioni correttive identificate;
- **Follow-up**: le modifiche apportate dall'autore vengono controllate per accertare la loro corretta implementazione.

Data l'inesperienza del team nell'attività di verifica, inizialmente è stata utilizzata la tecnica "walkthrough". Ciò ha permesso al gruppo di rilevare gli errori più comuni e acquisire le conoscenze necessarie per definire le liste di controllo ed eseguire verifiche più mirate. Le checklist sono raccolte nel *Piano di Qualifica v2.0.0*.

3.4.4 Analisi dinamica

L'analisi dinamica è un processo di verifica basato sull'osservazione del comportamento di un sistema software o di un suo componente in esecuzione. Spesso il termine "testing" viene utilizzato come sinonimo di analisi dinamica, poiché quest'ultima prevede la definizione di un insieme di prove (test), preferibilmente automatizzate e riproducibili. Le precondizioni necessarie per poter effettuare un test sono la configurazione dell'ambiente di esecuzione e la conoscenza del comportamento atteso (determinato dall'oracolo). Un oracolo è un metodo usato nella verifica del software per determinare se un test ha avuto successo o è fallito. L'obiettivo dei test è produrre una misura quanto più oggettiva della qualità del prodotto e, di conseguenza, devono essere eseguiti in parallelo all'attività di codifica. Come parte del processo di analisi dinamica, il team ha individuato le seguenti tipologie di test da eseguire:

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

La classificazione dettagliata dei test è disponibile nella sezione §2.2.5.

3.4.5 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Discord;
- Google Meet;
- GitHub.

3.5 Validazione

3.5.1 Scopo

Il processo di validazione ha lo scopo di determinare se le caratteristiche del software soddisfano i bisogni dell'utente e l'uso previsto. La validazione è una conferma finale, e consiste in una serie di attività volte a garantire che il prodotto soddisfi i requisiti specificati nell'*Analisi dei Requisiti v2.0.0*.

3.5.2 Implementazione del processo

L'esito positivo dei test di unità, di integrazione e di sistema rappresenta una precondizione necessaria per avviare il processo di convalida del software. La validazione viene eseguita alla presenza dell'azienda proponente. Il processo di validazione include le seguenti attività:

- **Revisione dei requisiti**: il team accerta che tutte le funzionalità richieste siano state implementate;
- **Collaudo**: il team esegue i test di accettazione per assicurarsi che il prodotto funzioni come previsto.

Una volta accertato che i requisiti sono stati implementati in maniera consistente, il prodotto viene convalidato dall'azienda proponente.

3.5.3 Test di accettazione

I test di accettazione accertano il soddisfacimento dei requisiti utente. Sono test black-box il cui obiettivo principale non è tanto quello di individuare difetti, quanto di confermare che il prodotto risponda alle reali necessità dell'utente finale. I test di accettazione prevedono l'esecuzione di una serie di casi di test, ognuno dei quali simula uno scenario di utilizzo del prodotto, il più possibile conforme alle condizioni d'uso reali. Si tratta di test formali che si concentrano sulla replica dei comportamenti degli utenti e, pertanto, richiedono che tutta l'applicazione sia attiva. Possono essere definiti già a partire dal capitolato.

3.5.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Zoom.

3.6 Revisione congiunta

3.6.1 Scopo

Le revisioni congiunte consistono in attività di valutazione dei risultati conseguiti dal gruppo da parte di un componente esterno.

3.6.2 Implementazione del processo

Le revisioni congiunte avvengono tra il gruppo e la *Proponente*$_G$ con l'obiettivo di verificare che i requisiti e la loro implementazione rispettino le aspettative. A tal fine, vengono organizzati degli incontri periodici durante i quali il team espone il lavoro svolto. Il gruppo propone un incontro tramite Gmail; la Proponente risponde indicando il giorno e l'ora più opportuni. Da qui il gruppo conferma e si impegna a inviare un link per la chiamata su Zoom.

Durante l'incontro, il team espone i risultati delle attività, interagendo con la *Proponente*$_G$ per ottenere un riscontro sull'operato o sui dubbi emersi.

Le riunioni vengono verbalizzate e i documenti elaborati vengono sottoposti all'approvazione della *Proponente*$_G$.

3.6.3 Revisioni tecniche congiunte

Durante le revisioni, il team e la *Proponente*$_G$ discutono i requisiti individuati. Inoltre, le riunioni vertono sull'approfondimento delle strategie di composizione del *prompt*$_G$ e sulla valutazione della correttezza delle query generate dagli *LLM*$_G$. Questo processo viene effettuato generando uno o più prompt; ciascun prompt viene fornito in input a *ChatGPT*$_G$ o altri modelli su chatLMsys per ottenere query SQL. L'attività di revisione è

accompagnata da analisi costruttive sulla correttezza, sintattica e semantica, delle query.

3.6.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Zoom;
- chatLMsys;
- ChatGPT;
- Gmail.

3.7 Audit

3.7.1 Scopo

Il processo di audit serve a determinare l'adesione ai requisiti, alla pianificazione e ai vincoli del progetto. Attuare il processo richiede solamente due parti interlocutrici, di cui una revisiona le attività e/o il prodotto dell'altra.

3.7.2 Implementazione del processo

Le revisioni sono pianificate e concordate, nel contenuto e nella data, da ciascuna delle parti coinvolte. Eventuali criticità individuate durante la revisione devono essere documentate e gestite, attuando il processo di risoluzione problemi (3.8). I risultati di una revisione e le azioni conseguenti devono essere stabiliti in comune accordo tra le due parti e opportunamente documentati.

3.7.3 Revisioni di project management

Le revisioni di project management sono attuate durante le riunioni interne, e registrate attraverso i corrispondenti verbali e tramite il consuntivo di periodo nel *Piano di Progetto*. Monitorano principalmente il corso delle attività svolte durante lo sprint in esame e individuano azioni correttive da attuare a breve o medio termine, in base all'urgenza dei problemi individuati.

3.7.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- LaTeX;
- Discord;
- Telegram;
- Google Meet.

3.8 Risoluzione dei problemi

3.8.1 Scopo

Il processo di risoluzione dei problemi mira ad analizzare e contrastare i problemi che possono insorgere durante lo sviluppo. Il gruppo esamina costantemente lo stato di esecuzione dei processi per individuare criticità e difetti. Una volta rilevato un problema, il gruppo si impegna ad arginarlo tempestivamente, documentando le cause e pianificando soluzioni per ridurre la probabilità che si ripresenti.

3.8.2 Implementazione del processo

All'insorgere di una criticità, i membri del gruppo devono segnalare prontamente il problema, tracciandolo nell'*ITS*$_G$ come ticket di tipo "bug". Inoltre, il problema deve essere comunicato al resto del team tramite i canali di comunicazione dedicati (sezione §4.1.7). Chi rileva il bug ha il compito di descrivere la natura del problema e aprire uno spazio di discussione su *GitHub*$_G$. Ogni problema viene classificato per categoria e priorità.

Spesso le criticità emergono durante le riunioni; in tal caso, verranno documentate in un verbale interno. Una volta individuato e segnalato un problema, questo viene preso in carico da o uno più membri del gruppo, i quali si occuperanno di risolverlo e di formalizzare la soluzione.

3.8.3 Risoluzione dei problemi

Se il problema è stato rilevato in un *branch*$_G$ già aperto, un membro del team può apportare direttamente una correzione (bug fix) che verrà poi integrata all'interno dell'ambiente condiviso.

In caso contrario, verrà aperto un branch dedicato per la risoluzione.

3.8.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Jira;
- Zoom;
- Discord;
- Telegram;
- Google Meet;
- GitHub.

4 Processi organizzativi

4.1 Gestione

4.1.1 Descrizione

Il processo di gestione include i task che vengono eseguiti dal *responsabile di progetto* per il coordinamento del progetto.

Tra le attività principali del processo di gestione si possono distinguere:

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

4.1.2 Pianificazione

Questa fase include l'assegnazione di ruoli e attività. Ogni attività è corredata da:

- Uno o più membri incaricati di svolgere l'attività;
- Data di inizio e fine;
- Priorità;
- Uno o più verificatori incaricati di valutare il risultato;
- Eventuali rischi associati all'attività.

Una volta pianificata un'attività, il *responsabile di progetto* e l'*amministratore* hanno il compito di aprire uno o più ticket su $Jira_G$, seguendo le linee guida riportate nella sezione §3.2.

Strumenti Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Jira;
- Google Sheets.

4.1.3 Esecuzione e controllo

Il *responsabile di progetto* deve garantire l'esecuzione e il mantenimento del piano definito nella fase precedente, affrontando e risolvendo eventuali problemi emersi durante l'avanzamento del processo. La risoluzione dei problemi può richiedere modifiche alla pianificazione. Per assicurare che l'impatto di tali modifiche sia adeguatamente controllato, è fondamentale documentare i problemi riscontrati con le relative soluzioni. Inoltre, il *responsabile di progetto* ha il compito di monitorare l'esecuzione del processo, fornendo report sia per uso interno che esterno.

Strumenti Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Jira;
- GitHub;
- Google Sheets.

4.1.4 Valutazione e approvazione

La revisione di un'attività è effettuata dal *verificatore* incaricato di esaminare la pull request prima della sua integrazione nell'ambiente condiviso. Dopo aver accettato, o rifiutato, le modifiche al codice o alla documentazione, il *verificatore* deve richiedere l'approvazione finale del *responsabile di progetto*. In questo modo, il team garantisce la validità dei risultati delle attività completate durante il processo. In particolare, il gruppo deve assicurare che i requisiti del software siano soddisfatti e che la documentazione sia completa e corretta.

Strumenti Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- GitHub.

4.1.5 Ruoli

Il progetto didattico prevede l'assegnazione di sei ruoli distinti, con una rotazione costante e bilanciata che va considerata nella pianificazione delle attività.

4.1.5.1 Responsabile

Il *responsabile di progetto* coordina le attività interne al gruppo e gestisce la comunicazione con la Proponente e i Committenti.

I compiti assegnati al *responsabile di progetto* sono i seguenti:

- Pianificazione delle attività;
- Stesura del preventivo dello sprint;
- Redazione e miglioramento continuo del *Piano di Progetto*;
- Tracciamento e monitoraggio delle attività del gruppo;
- Valutazione e gestione dei rischi;
- Preparazione dell'ordine del giorno e moderazione delle riunioni;
- Sviluppo del consuntivo dello sprint.

4.1.5.2 Amministratore

L'*amministratore* è incaricato della configurazione e della manutenzione dell'ambiente di sviluppo condiviso.

I compiti assegnati all'*amministratore* sono i seguenti:

- Svolgimento delle attività legate al processo di "configuration management" (sezione §3.2);
- Stesura delle *Norme di Progetto* (non come unico *redattore*, ma come supervisore);
- Misurazione delle metriche di qualità e aggiornamento del *Piano di Qualifica*.

4.1.5.3 Analista

Il ruolo dell'*analista* è focalizzato sull'ingegneria dei requisiti.

I compiti assegnati all'*analista* sono i seguenti:

- Svolgimento delle attività descritte nella sezione §2.2.2;
- Redazione del documento di *Analisi dei Requisiti*.

4.1.5.4 Progettista

Il *progettista* definisce soluzioni e stili architetturali per lo sviluppo del prodotto.

I compiti assegnati al *progettista* sono i seguenti:

- Svolgimento delle attività descritte nella sezione §2.2.3;
- Redazione del documento Specifica Tecnica.

4.1.5.5 Programmatore

Il *programmatore* implementa l'architettura delineata dal *progettista*. Inoltre, si occupa di definire test di unità e di integrazione per la verifica del prodotto software.

I compiti assegnati al *programmatore* sono i seguenti:

- Svolgimento delle attività descritte nella sezione §2.2.4 e nella sezione §2.2.5.

4.1.5.6 Verificatore

Il *verificatore* si occupa della revisione delle modifiche apportate dal gruppo, assicurandosi che i risultati delle attività soddisfino gli standard di qualità e di funzionamento previsti.

I compiti assegnati al *verificatore* sono i seguenti:

- Accertamento della qualità del codice e della documentazione;
- Approvazione di un avanzamento di versione (escluso il rilascio).

4.1.6 Gestione dei rischi

La gestione dei rischi è un'attività svolta dal *responsabile di progetto*, con il supporto dell'*amministratore* per quanto riguarda i rischi legati all'infrastruttura. Questa attività è inclusa nel *Piano di Progetto v2.0.0* e viene eseguita alla fine di ogni sprint come parte del consuntivo di periodo. La pratica di gestione dei rischi alimenta e influenza l'analisi dei rischi, anch'essa riportata nel *Piano di Progetto v2.0.0*.

4.1.6.1 Rischi: Notazione

I rischi vengono identificati in modo univoco secondo questa notazione:

$$R[Tipo][Numero]$$

dove:

- **R**: indica la parola "rischio";
- **Tipo**: indica il tipo di rischio:
  - **O**: rischio organizzativo;
  - **T**: rischio tecnologico;
  - **P**: rischio di natura personale.
- **Numero**: è un numero progressivo che identifica in modo univoco il rischio per ogni tipologia.

4.1.6.2 Rischi: Didascalia

I rischi sono definiti come segue:

- **Probabilità**: stima di occorrenza del rischio (Alta, Media, Bassa);
- **Grado di criticità**: impatto del rischio (Alto, Medio, Basso);
- **Descrizione**: descrizione del rischio;
- **Strategie di rilevamento**: metodi per anticipare l'insorgenza di situazioni problematiche;
- **Contromisure**: misure di mitigazione da applicare qualora il rischio si manifesti.

4.1.7 Comunicazione

Il gruppo adotta canali di comunicazione differenti per la comunicazione interna (tra i membri del gruppo) e quella esterna (con la Proponente e i Committenti). Di seguito sono illustrate le modalità di comunicazione e le applicazioni utilizzate.

4.1.7.1 Comunicazione interna

La comunicazione tra i membri del gruppo avviene attraverso le seguenti modalità di interazione:

- **Telegram** (comunicazione asincrona): utilizzato per comunicazioni di interesse generale e di breve contenuto. È possibile creare gruppi di discussione per argomenti specifici. I messaggi più importanti vengono fissati all'interno della rispettiva chat;
- **Slack** (comunicazione asincrona): utilizzato per inviare promemoria ai verificatori riguardo alla revisione delle pull request;
- **Discord** (comunicazione sincrona): utilizzato per chiamate di gruppo, formali o informali, e per la comunicazione interna tra membri del team che ricoprono lo stesso ruolo. È possibile creare canali di testo e vocali per argomenti specifici. Inoltre, il gruppo utilizza Discord come piattaforma di supporto per la "pair programming", disponibile da remoto tramite condivisione schermo o per mezzo dell'estensione Live Share di Visual Studio Code;
- **Google Meet** (comunicazione sincrona): fallback per le chiamate di gruppo, in caso di problemi con Discord;
- **Incontri in presenza** (comunicazione sincrona): il gruppo si riunisce in presenza con cadenza mensile (o bisettimanale a ridosso delle revisioni di avanzamento).

Questi strumenti devono essere integrati con un *Issue Tracking System*$_G$, poiché le decisioni prese durante gli incontri, a meno che non siano documentate in un verbale, sono più difficili da tracciare e recuperare in futuro.

Strumenti Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Telegram;
- Slack;
- Discord;
- Google Meet.

4.1.7.2 Comunicazione esterna

La comunicazione con la Proponente e i Committenti avviene attraverso le seguenti modalità di interazione:

- **Gmail** (comunicazione asincrona): utilizzata per comunicazioni formali. La comunicazione esterna è gestita dal *responsabile di progetto*, attraverso il recapito di posta elettronica del gruppo argo.unipd@gmail.com;
- **Zoom** (comunicazione sincrona): utilizzato per incontri formali.

Strumenti Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Gmail;
- Zoom.

4.2 Miglioramento

4.2.1 Scopo

Il processo di miglioramento si occupa di stabilire, misurare, controllare e migliorare i processi di ciclo di vita del software.

4.2.2 Attività

Il processo si compone delle seguenti attività:

- Definizione del processo;
- Valutazione del processo;
- Miglioramento del processo.

4.2.3 Definizione del processo

Vengono riconosciuti i processi da implementare e ne vengono documentate le norme stabilite dal Way of Working. Quando opportuno, il team implementa un meccanismo di controllo per monitorare e migliorare il processo.

4.2.4 Valutazione del proceso

Il gruppo pianifica ed esegue revisioni dei processi a intervalli regolari corrispondenti alle iterazioni di progetto, o più strettamente quando il processo richiede controllo costante (ad esempio durante le prime fasi di implementazione).

Le revisioni hanno l'obiettivo di assicurare l'efficacia del processo, individuando gli aspetti migliorabili per l'avvicinamento allo stato dell'arte.

4.2.5 Miglioramento del processo

Il gruppo effettua miglioramenti al processo evidenziati e riconosciuti durante la valutazione. Le norme inerenti al processo vengono aggiornate in modo da comprendere gli aspetti individuati.

Informazioni di storico, tecniche o di valutazione vengono collettivamente raccolte e analizzate per riconoscere punti di forza e aspetti deboli dei processi impiegati. Il risultato dell'analisi viene impiegato come feedback per il miglioramento del processo.

4.2.6 Strumenti

- Jira;
- GitHub;
- LaTeX;
- Discord;

- Telegram.

4.3 Formazione

4.3.1 Scopo

Il processo di formazione ha lo scopo di fornire al team le competenze necessarie per svolgere le attività del progetto e, soprattutto, di mantenere aggiornate le conoscenze all'interno del gruppo. La fornitura, lo sviluppo e la manutenzione dei prodotti software dipendono in gran parte da personale esperto e qualificato.

4.3.2 Implementazione del processo

Il team deve condurre una revisione dei requisiti del progetto per identificare le risorse e le competenze richieste. Inoltre, è necessario sviluppare un piano di formazione che includa le seguenti attività:

- **Formazione individuale**: prevede lo studio della documentazione relativa alle tecnologie e agli strumenti utilizzati. Alla formazione teorica segue un approccio pratico di tipo "learning by doing";
- **Workshop** (formazione collaborativa): sessioni in cui i membri più esperti condividono le loro conoscenze per uniformare le competenze all'interno del gruppo.

4.3.3 Materiale di formazione

Il materiale di formazione utilizzato dal gruppo comprende:

- Documentazione ufficiale delle tecnologie o degli strumenti utilizzati;
- Forum e community online;
- Manuali di formazione redatti dal team;
- Presentazioni multimediali;
- Esempi pratici (es.: programmi già sviluppati).

Il materiale di formazione è organizzato in una sezione dedicata su Google Drive. Per le risorse web, viene fornito l'indirizzo di destinazione delle pagine.

4.3.4 Strumenti

Gli strumenti elencati in seguito sono approfonditi nella sezione Strumenti (5).

- Discord;
- Google Meet;
- Google Drive;
- Google Docs;

- Google Slides;
- GitHub;
- Visual Studio Code;
- Jira.

5 Strumenti

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

5.1 Canva

https://www.canva.com (Ultimo accesso: 2024-09-20)

Strumento di progettazione grafica utilizzato per la creazione di loghi e immagini.

5.2 chatLMsys

https://chat.lmsys.org/ (Ultimo accesso: 2024-09-20)

Sito web per benchmarking di LLM_G.

5.3 Colour Contrast Analyzer

https://www.tpgi.com/color-contrast-checker (Ultimo accesso: 2024-09-20)

Strumento utilizzato per la scelta della palette di colori, in conformità con le linee guida WCAG 2.1 sull'accessibilità del rapporto di contrasto.

5.4 Diagrams.net

https://app.diagrams.net (Ultimo accesso: 2024-09-20)

Strumento utilizzato per la creazione dei diagrammi dei casi d'uso, di attività e di sequenza in UML.

5.5 Discord

https://discord.com (Ultimo accesso: 2024-09-20)

Piattaforma utilizzata per lo svolgimento di riunioni formali e informali. Il team ha configurato un server con un canale dedicato ai messaggi di testo e diverse sale virtuali riservate alle conversazioni vocali.

5.6 Docker

https://www.docker.com (Ultimo accesso: 2024-09-20)

Piattaforma software utilizzata per sviluppare, testare e distribuire applicazioni in contenitori, consentendo di creare un ambiente isolato per l'esecuzione di processi.

5.7 Git

https//git-scm.org (Ultimo accesso: 2024-09-20)

Version Control System$_G$ distribuito e open source.

5.8 GitHub

https://github.com (Ultimo accesso: 2024-09-20)

Piattaforma di hosting per lo sviluppo software collaborativo. GitHub offre strumenti per il controllo di versione, tracciamento delle attività, verifica del codice, integrazione delle modifiche e distribuzione del codice.

5.9 Google Calendar

https://calendar.google.com (Ultimo accesso: 2024-09-20)

Applicazione utilizzata per organizzare gli eventi e le riunioni formali, sincronizzando in automatico le informazioni su tutti i dispositivi connessi.

5.10 Google Docs

https://docs.google.com/document (Ultimo accesso: 2024-09-20)

Applicazione utilizzata per elaborare documenti online collaborando con altri utenti in tempo reale.

5.11 Google Drawings

https://docs.google.com/drawings (Ultimo accesso: 2024-09-20)

Software utilizzato per la creazione di diagrammi e disegni di natura tecnica.

5.12 Google Forms

https://docs.google.com/forms (Ultimo accesso: 2024-09-20)

Applicazione utilizzata per la creazione di questionari di valutazione degli sprint in combinazione con Google Sheets.

5.13 Google Gmail

https://mail.google.com (Ultimo accesso: 2024-09-20)

Servizio di posta elettronica utilizzato per la comunicazione via email con i Committenti e il Proponente.

5.14 Google Sheets

https://docs.google.com/spreadsheets (Ultimo accesso: 2024-09-20)

Applicazione utilizzata per creare e gestire fogli di calcolo online, agevolando la collaborazione in tempo reale e la generazione di grafici.

5.15 Google Slides

https://docs.google.com/presentation (Ultimo accesso: 2024-09-20)

Applicazione utilizzata per la creazione delle slide per i diari di bordo e per le presentazioni relative alle revisioni di avanzamento.

5.16 Jira

https://www.atlassian.com/it/software/jira (Ultimo accesso: 2024-09-20)

Software utilizzato per il monitoraggio delle attività e la gestione dei progetti sviluppati con metodologie agili.

5.17 LaTeX

https://www.latex-project.org (Ultimo accesso: 2024-09-20)

Linguaggio di markup utilizzato per la stesura di documenti tecnici e scientifici; LaTeX garantisce una struttura e una formattazione flessibili e professionali.

5.18 Latexmk

https://pypi.org/project/latexmk.py (Ultimo accesso: 2024-09-20)

Strumento utilizzato per automatizzare il processo di compilazione di un documento scritto in LaTeX. Latexmk deriva dall'utilità generica "make" ed è in grado di determinare in automatico le dipendenze. Inoltre, risolve i riferimenti incrociati ed esegue nuovamente LaTeX ogni volta che un file sorgente viene aggiornato.

5.19 PDF24 Tools

https://tools.pdf24.org/it/svg-in-pdf (Ultimo accesso: 2024-09-20)

Strumento utilizzato per convertire file SVG in PDF.

5.20 Slack

https://slack.com (Ultimo accesso: 2024-09-20)

Software utilizzato in combinazione con GitHub per inviare promemoria a intervalli di tempo regolari. I promemoria sono indirizzati ai verificatori a cui sono state assegnate pull request pronte per la revisione.

5.21 StarUML

https://staruml.io (Ultimo accesso: 2024-09-20)

Software utilizzato per la creazione dei diagrammi delle classi in UML.

5.22 Table Convert Online

https://tableconvert.com/it/csv-to-latex (Ultimo accesso: 2024-09-20)

Strumento utilizzato per convertire file csv_G in $LaTeX_G$.

5.23 Telegram

https://web.telegram.org (Ultimo accesso: 2024-09-20)

Applicazione di messaggistica utilizzata per creare gruppi tematici, semplificando la comunicazione e l'organizzazione all'interno del team.

5.24 Visual Studio Code

https://code.visualstudio.com (Ultimo accesso: 2024-09-20)

Editor di codice sorgente utilizzato per lo sviluppo software. Visual Studio Code offre un'ampia gamma di funzionalità, tra cui l'integrazione con Git e Copilot, strumenti di debug ed estensioni che permettono di personalizzare l'ambiente di sviluppo.