Norme di Progetto

Gruppo Argo — ***Progetto ChatSQL***

Informazioni sul documento	
Versione	0.0.15
Approvazione	TODO
Uso	Interno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo Argo

UNIVERSITÀ DEGLI STUDI DI PADOVA

Registro delle modifiche

Ver.	Data	Redazione	Verifica	Descrizione
0.0.15	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 tutto lo svolgimento del progetto didattico. Poiché il way of working è definito incrementalmente durante il corso del progetto, questo documento non è da considerarsi un testo definitivo o completo.

1.2 Scopo del prodotto

1.3 Glossario

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

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato C9 - ChatSQL:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9.pdf
    (Ultimo accesso: 2024-07-02);
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9p.pdf
    (Ultimo accesso: 2024-07-02).
- Standard ISO/IEC 12207:1995:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf.

1.4.2 Riferimenti informativi

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

- Documentazione ufficiale di Jira:
  https://confluence.atlassian.com/jira
  (Ultimo accesso: 2024-05-15);
- Continuous integration:
  https://it.wikipedia.org/wiki/Integrazione_continua
  (Ultimo accesso: 2024-05-15);
- Continuous delivery:
  https://www.atlassian.com/it/continuous-delivery
  (Ultimo accesso: 2024-05-15);
- Slide T2 - Corso di Ingegneria del Software - Processi di ciclo di vita del Software:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T2.pdf
  (Ultimo accesso: 2024-04-11);
- Slide T3 - Corso di Ingegneria del Software - Modelli di sviluppo del Software:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T3.pdf
  (Ultimo accesso: 2024-04-11);
- Slide T4 - Corso di Ingegneria del Software - Gestione di Progetto:
  https://www.math.unipd.it/ tullio/IS-1/2023/Dispense/T4.pdf
  (Ultimo accesso: 2024-04-11);
- Normative e standard:
  https://www.humanwareonline.com/project-management/center/differenze-tra-normative-standard
  (Ultimo accesso: 2024-05-02);
- Esempi di metriche di prodotto:
  http://www.colonese.it/00-Manuali_Pubblicatii/07-ISO-IEC9126_v2.pdf
  (Ultimo accesso: 2024-07-15);
- *Piano di Progetto v0.1.9*;
- *Piano di Qualifica v0.2.0*;
- *Analisi dei Requisiti v1.0.0*;
- *Glossario 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.
- Verbali esterni:
  - 2024-04-09;
  - 2024-05-06;
  - 2024-05-22;
  - 2024-06-07;
  - 2024-07-09.

2 Processi primari

2.1 Fornitura

2.1.1 Descrizione

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

- Selezione e studio fattibilità;
- Candidatura;
- Pianificazione;
- Esecuzione e controllo;
- Revisione e valutazione;
- Consegna e completamento.

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

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

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

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

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

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

2.1.1.5 Revisione e valutazione Il Fornitore coordina la revisione interna ed esegue verifica e validazione secondo le norme definite in 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 sugli avanzamenti prodotti;
- Presentazioni dell'avanzamento del prodotto software.

L'approccio che la Proponente mantiene durante il corso del progetto, in quanto *didattico*, è quella di mantenere un profilo 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 corrisponde a un Verbale Esterno prodotto che ne riassume i punti chiave della discussione.

2.1.3 Documentazione fornita

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

2.1.3.1 Piano di Progetto: Il *Piano di Progetto* è il documento che tratta della gestione e dell'organizzazione del gruppo. Tratta dell'analisi e la gestione dei rischi, valuta lo stato del progetto rappresentando la divisione delle ore per $sprint_G$, il preventivo e il consuntivo orari ed economici e definisce una retrospettiva dello $sprint_G$ precedente e la pianificazione degli $sprint_G$ successivi. 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 dei rischi sulle risorse del progetto, le strategie di rilevamento e di mitigazione per contrastarli. L'analisi dei rischi viene suddivisa in sezioni 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, definendone la struttura e i punti di forza;
4. **Stima temporale del progetto:** indica la stima delle date delle revisioni alle quali il gruppo si deve sottoporre. Viene indicata l'ultima data di aggiornamento della sezione;
5. **Stima dei costi:** serve a dare una stima preliminare sui costi del progetto. Viene aggiornata ad ogni $sprint_G$;
6. **Pianificazione:** serve a pianificare lo $sprint_G$ definendo gli obbiettivi che il gruppo vuole raggiungere. Viene disposta una sezione di pianificazione per ogni $sprint_G$;
7. **Preventivo:** dispone il preventivo per lo $sprint_G$. Ogni $sprint_G$ ha sezione di preventivo. 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 totale, ossia il preventivo economico per lo $sprint_G$.

8. **Consuntivo:** dispone il consuntivo per lo *sprint*$_G$. Ogni *sprint*$_G$ ha sezione di 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*$_G$.
    - 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*$_G$;
    - Retrospettiva: indica i risultati del questionario che ogni membro deve compilare al termine di ogni *sprint*$_G$ per valutarne l'andamento. Oltre a ciò, vengono raccolte delle considerazioni sullo *sprint*$_G$ come la presenza di rischi;
    - Aggiornamento pianificazione e preventivo: definisce le azioni migliorative per il prossimo *sprint*$_G$ e una prima pianificazione dei macro obiettivi da esaminare nella pianificazione dello *sprint*$_G$ successivo. Delinea inoltre la gestione dei rischi, ossia i rischi rilevati durante lo *sprint*$_G$, 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 si tratta della specifica degli obbiettivi quantitativi di qualità di prodotto e processo. Delinea quindi un'insieme di indici di vautazione 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 righe delle metriche con le colonne: 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 agli standard di qualità del cliente.
    - Qualità per obiettivo: misura la qualità dei processi primari, di supporto e organizzativi dello sviluppo software.
3. **Verifica:** sezione dedicata alla verifica e al collaudo del software. Misura a rilevare, correggere e prevenire i possibili errori del software. È divisa in sezioni in base alla tipologia di test. Le sezioni sono dotate di tabelle che descrivono le righe dei test con le colonne: ID, descrizione e stato del test. I test sono così divisi:
    - 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 descrive le righe di checklist con le colonne: titolo e descrizione.
4. **Cruscotto di valutazione delle qualità:** sezione dedicata alla dimostrazione del rispetto delle norme. Le sottosezioni sono norme alle quali è stata data 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 descrive l'andamento. Le metriche nel cruscotto di qualità vengono aggiornate periodicamente per aggiornarle 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 prima fase di revisione di avanzamento: la

Requirements and Technology Baseline. Vengono riportati i *repository*$_G$ di documentazione e di codice sorgente, assieme all'indirizzo dove il Proof of Concept. Infine viene dato l'aggiornamento impegni con la stima del preventivo a finire aggiornato del progetto.

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 prodotta.

2.1.4 Strumenti

Gli strumenti impiegati nel processo di fornitura sono:

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

2.1.4.1 Dashboard Google Sheets

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

- Un foglio contenente le variabili globali del *Piano di Progetto*, inclusi il costo orario di ciascun ruolo e il budget totale;
- Un foglio nascosto per ciascun periodo, che include le risposte al questionario di valutazione dello sprint;
- Un foglio per ogni sprint con le seguenti informazioni:
  - Numero dello *sprint*$_G$;
  - Date di inizio e termine dello *sprint*$_G$;
  - Tabella di assegnazione delle ore produttive per ciascun membro del team, accumulate in totali per persona e per ruolo;
  - Distribuzione delle ore per ruolo, sotto forma di donut chart;

- Distribuzione delle ore per la coppia risorsa-ruolo, sotto forma di grafico a barre sovrapposte (così da poter assumere più ruoli per sprint);
- Preventivo economico dello *sprint*$_G$;
- Tabella riassuntiva con ore e budget spesi e restanti;
- Pie chart con la stima delle ore spese sul totale;
- Pie chart con la stima del budget sul totale;
- Ore rimanenti per la coppia risorsa-ruolo.

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

- Preventivo;
- Consuntivo.

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

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

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

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

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

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

Nella sezione rendicontazione ore, il responsabile deve inserire:

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

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

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

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

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

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

  - Delta ore preventivo - consuntivo: differenza tra le ore preventivate e quelle effettivamente spese;
  - Costo (in €);
  - Delta costo preventivo - consuntivo: differenza tra il costo preventivato e quello effettivo.
  - Ore e budget spesi negli *sprint*$_G$ precedenti;
  - Ore e budget restanti.

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

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

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

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

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

2.2 Sviluppo

2.2.1 Descrizione

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

- Analisi dei requisiti;

- *Progettazione*$_G$;
- *Codifica*$_G$ e *testing*$_G$.

2.2.2 Analisi dei Requisiti

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

- 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 UC è divisa nella seguente struttura:

- UCN - Nome UC;
- 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 UC va tenuto conto delle seguentio considerazioni:

- Le precondizioni dello UC devono essere condizioni necessarie per arrivare alla situazione che si presenta nello UC.
- Le precondizioni vengono poste al passato per identificare la conclusione di un UC, possono includere altri UC;
- Le postcondizioni rappresentano 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 ad un UC viene riportato in precondizioni, postcondizioni, scenario principale, scenario alternativo, inclusione ed estensione;

- I sottocasi d'uso di uno UC vengono inseriti sotto lo UC principale nello stesso file;
- I sottocasi sono sempre inclusioni del caso d'uso "padre";
- I sottocasi vengono riferiti con un punto dopo il padre. Ad esempio un caso d'uso potrebbe essere UC1 e il sottocaso UC1.1;
- Per gli errori si visualizza quasi sempre un messaggio. Quindi la postcondizione finale e la fine dello scenario principale sarà quasi sempre "Viene visualizzato un messaggio con i dettagli dell'errore";
- 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*$_G$.

Data tale classificazione, i requisiti vengono identificati da un indice univoco con la seguente struttura:

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

Dove *R* sta per Requisito, *Tipologia* è la sigla associata alla categoria del requisito, *Importanza* è la sigla per il valore di importanza 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, per cui ciascuna fonte di un requisito viene indicata assieme alla sua descrizione.

2.2.3 Progettazione

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

2.2.4 Codifica e testing

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

2.2.4.2 Elementi comuni di stile

- La lingua da utilizzare nella nomenclatura di termini all'interno del codice è inglese, sono esenti commenti o contenuti di testo che necessitano la lingua italiana;
- Sezioni incomplete vanno indicate con un commento contenente "TODO", sezioni di codice non funzionanti o da rivedere vanno indicate con un commento contenente "FIXME" per una individuazione più semplice e una ricerca agevolata all'interno del codice.

2.2.4.3 Python

- **Tab size:** 4;
- **Nome delle variabili:** minuscolo;
- **Nome dei metodi:** camelCase;

2.2.4.4 JavaScript/TypeScript TODO

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

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

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

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

- Titolo;

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

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

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

3.1.2 Lista documenti

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

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

3.1.3 Ciclo di vita

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

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

3.1.4 Ambiente di lavoro

3.1.4.1 *LaTeX*$_g$ 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 del caso di riga associata al rilascio);
  - Verificatore della modifica (coincide con il Responsabile nel caso di riga associata al rilascio);
  - Descrizione della modifica.
- Indice dei contenuti;

3.1.5.1 Verbali

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

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

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

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

3.1.6 Stile

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

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

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

3.1.6.2 Formattazione testo

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

3.1.7 Strumenti

Gli strumenti impiegati nel processo di documentazione sono:

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

3.2 Gestione della configurazione

3.2.1 Scopo

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

3.2.2 Descrizione

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

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

3.2.3 Issue tracking system (ITS)

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

3.2.3.1 Ticket

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

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

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

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

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

3.2.4 Automazione chiusura ticket

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

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

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

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

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

3.2.5 Pull request

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

- **main:** ramo di produzione;

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

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

3.2.5.1 Workflow

Il team adotta due tipi di workflow:

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

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

3.2.5.2 Apertura pull request

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

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

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

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

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

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

3.2.5.3 Verifica pull request

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

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

Il verificatore può applicare i seguenti filtri:

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

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

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

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

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

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

3.2.5.4 Chiusura pull request

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

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

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

3.2.6 Versionamento

Il gruppo mantiene un versionamento per il codice e la documentazione nel seguente formato:

$$X.Y.Z$$

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

Y Avanza ad ogni verifica completa del documento;

Z Avanza ad ogni modifica verificata di un documento.

3.2.7 Repository

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

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

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

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

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

Di seguito è riportata la struttura del repository:

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

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

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

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

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

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.

3.3 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 v0.2.0*;
- **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 v0.1.9*).

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 v0.1.9*).

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 v0.1.9*).

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 v0.1.9*;
- **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 v1.0.0*;
- **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**:

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

Efficienza delle contromisure nei rischi

- **Notazione specifica**: M.PC.12;
- **Nome**: Efficienza 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**:

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

  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 v1.0.0*;
- **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 v1.0.0*;

- **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 v1.0.0*;
- **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.

Accuratezza della risposta

- **Notazione specifica**: M.PD.13;
- **Nome**: Accuratezza della risposta;
- **Descrizione**: Questa metrica misura la correttezza dei risultati forniti dal sistema rispetto alle esigenze specificate;
- **Caratteristiche**: Funzionalità, adeguatezza;
- **Motivo**: Garantire che il sistema fornisca risposte in linea con le aspettative;
- **Misurazione**:

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

  Dove:

  - R_c: Numero di risposte corrette;
  - N_r: Numero totale di richieste.

  Il team eseguirà i test definiti con la *Proponente*$_G$ durante le riunioni esterne.

Linee medie di codice per metodo

- **Notazione specifica**: M.PD.14;
- **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 con strumenti di analisi statica.

Complessità ciclomatica

- **Notazione specifica**: M.PD.15;
- **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.

Accoppiamento delle classi

- **Notazione specifica**: M.PD.16;
- **Nome**: Accoppiamento delle classi;
- **Descrizione**: Questa metrica misura il grado di dipendenza tra le classi del sistema;
- **Caratteristiche**: Manutenibilità;
- **Motivo**: Valutare la progettazione del software. Un accoppiamento elevato indica un prodotto difficile da riutilizzare e mantenere a causa delle sue numerose interdipendenze;
- **Misurazione**: Il calcolo viene effettuato con strumenti di analisi statica.

Indice di manutenibilità

- **Notazione specifica**: M.PD.17;
- **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 strumenti di analisi statica.

Percentuale di test superati

- **Notazione specifica**: M.PD.18;
- **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.19;
- **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.20;
- **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.21;
- **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 v0.2.0*. 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 (uno sviluppatore 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 v0.2.0*.

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**.

3.4.5 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 o componenti 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 o componenti 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.

3.4.6 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 componenti 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 componenti mancanti, consentendo di integrare fin da subito le funzionalità di "alto livello".

3.4.7 Test di sistema

I test di sistema verificano il comportamento dinamico dell'intero sistema rispetto ai requisiti specificati nel documento di *Analisi dei Requisiti v1.0.0*. Questa fase della verifica ha inizio al completamento dei test di unità e di integrazione, ed è 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.

3.4.8 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.

3.4.9 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.

3.5 Validazione

TODO

4 Processi organizzativi

4.1 Gestione

4.1.1 Descrizione

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

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

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

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

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

4.1.2 Ruoli

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

4.1.2.1 Responsabile Il *Responsabile di Progetto* è il ruolo che coordina le attività dell'intero gruppo internamente e gestisce i contatti con la Proponente e i Committenti come riferimento unico per tutto il gruppo.
La responsabilità del ruolo sono:

- Pianificazione dello sprint (task);
- Sviluppo del preventivo dello sprint;
- Redazione e cura del *Piano di Progetto*;
- Controllo delle attività del gruppo;
- Valutazione e gestione dei rischi;
- Preparazione dell'ordine del giorno e moderazione delle riunioni;
- Sviluppo del consuntivo dello sprint.

4.1.2.2 Amministratore L'*Amministratore* è il ruolo legato alla gestione della configurazione e manutenzione dell'ambiente di sviluppo comune.
Le responsabilità del ruolo sono:

- Attività legate al processo di gestione della configurazione;
- Cura delle *Norme di Progetto* (non come unico redattore, ma gestore delle modifiche da attuare);
- Misurazione delle metriche di qualità e aggiornamento del *Piano di Qualifica*;

4.1.2.3 Analista L'*Analista* è il ruolo legato all'attività di analisi, del processo di sviluppo. Le responsabilità del ruolo sono:

- Attività descritte in 2.2.2
- Redazione e cura dell'*Analisi dei Requisiti*.

4.1.2.4 Progettista Il *Progettista* definisce soluzioni architetturali e implementative per lo sviluppo del prodotto.
Le responsabilità del ruolo sono:

- Attività descritte in 2.2.3;

4.1.2.5 Programmatore Il *Programmatore* implementa l'architettura definita producendo codice che ne soddisfa le necessità. Inoltre si occupa di implementare test di unità e integrazione per la verifica del codice scritto.
Le responsabilità del ruolo sono:

- Attività descritte in 2.2.4.

4.1.2.6 Verificatore Il *Verificatore* controlla il risultato del lavoro degli altri ruoli accertando la qualità e il funzionamento del risultato delle attività.
Le responsabilità del ruolo sono:

- TODO

4.1.3 Comunicazione

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

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

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: 2024-07-14)

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

5.2 Colour Contrast Analyzer

https://www.tpgi.com/color-contrast-checker (Ultimo accesso: 2024-07-14)

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

5.3 Diagrams.net

https://app.diagrams.net (Ultimo accesso: 2024-07-14)

Strumento utilizzato per la creazione dei diagrammi dei casi d'uso in UML.

5.4 Discord

https://discord.com (Ultimo accesso: 2024-07-18)

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.

5.5 Docker

https://www.docker.com (Ultimo accesso: 2024-05-18)

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

5.6 GitHub

https://github.com (Ultimo accesso: 2024-07-18)

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.

5.7 Google Calendar

https://calendar.google.com (Ultimo accesso: 2024-07-18)

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

5.8 Google Docs

https://docs.google.com/document (Ultimo accesso: 2024-07-18)

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

5.9 Google Drawings

https://docs.google.com/drawings (Ultimo accesso: 2024-07-18)

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

5.10 Google Forms

https://docs.google.com/forms (Ultimo accesso: 2024-07-18)

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

5.11 Google Gmail

https://mail.google.com (Ultimo accesso: 2024-07-18)

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

5.12 Google Sheets

https://docs.google.com/spreadsheets (Ultimo accesso: 2024-07-18)

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

5.13 Google Slides

https://docs.google.com/presentation (Ultimo accesso: 2024-07-18)

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

5.14 Jira

https://www.atlassian.com/it/software/jira (Ultimo accesso: 2024-07-18)

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

5.15 LaTeX

https://www.latex-project.org (Ultimo accesso: 2024-05-18)

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

5.16 Latexmk

https://pypi.org/project/latexmk.py (Ultimo accesso: 2024-05-18)

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.

5.17 Slack

https://slack.com (Ultimo accesso: 2024-07-18)

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.

5.18 Telegram

https://web.telegram.org (Ultimo accesso: 2024-07-18)

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

5.19 Visual Studio Code

https://code.visualstudio.com (Ultimo accesso: 2024-05-18)

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.