Norme di Progetto

RAMtastic6

11 giugno 2024

email: ramtastic6@gmail.com

Informazioni sul documento

Versione:	2.0.0
Redattori:	Visentin S. Basso L. Tonietto F. Zambon M. Brotto D. Zaupa R.
Verificatori:	Tonietto F. Brotto D. Zambon M. Basso L. Zaupa R. Visentin S.
Destinatari:	T. Vardanega, R. Cardin, Imola Informatica
Uso:	Interno

Registro dei Cambiamenti - Changelog

Versione	Data	Autore	Verificatore	Dettaglio
v 2.0.0	2024-06-11	Tonietto F.	Tonietto F.	Approvazione Documento.
v 1.4.0	2024-06-08	Tonietto F.	Zambon M.	Ampliate le sezioni 4.3, 4.4 e 4.5 relative alla descrizione dei ruoli di "Amministratore di Progetto", "Analista" e "Progettista".
v 1.3.0	2024-05-20	Brotto D.	Tonietto F.	Aggiunta alla sezione "*Fornitura*$_G$"(2.2) i documenti "Analisi dei requisiti", "Lettera di presentazione", "*Specifica tecnica*$_G$" e "Manuale utente". Redatta la sezione "Progettazione"(2.3.2) , la sezione "*Codifica*$_G$"(2.3.3), la sezione "Testing"(3.4.5) e la sezione "Strumenti"(3.4.6).
v 1.2.0	2024-05-20	Brotto D.	Zambon M.	Aggiunta sezione "scopo e struttura *repository*$_G$ "EasyMeal".
v 1.1.1	2024-05-20	Basso L.	Zambon M.	Aggiornamento della sezione 3.2.4.1; inserita la convenzione per dare il nome ai *ticket*$_G$ inerenti alla *codifica*$_G$ e verifica del codice.
v 1.1.0	2024-05-15	Zaupa R.	Basso L.	Aggiornamento della sezione "Controllo di flusso" (3.2.4), inserito il flusso per la codifica e per la verifica del codice. Aggiornata la sezione "*Codifica*$_G$" (2.3.3). Inserite sottosezioni "Strumenti" per la "Documentazione" (3.1.8) e "*Controllo di Configurazione*$_G$".
v 1.0.1	2024-05-13	Zaupa R.	Basso L.	Inserimento sottosezioni "Strumenti" per i *processi primari*$_G$ di "*Fornitura*$_G$" (2.2.5), e "Sviluppo": "Analisi dei Requisiti" (2.3.1.4), "Progettazione" (2.3.2.1), "*Codifica*$_G$" (2.3.3.1)
v 1.0.0	2024-04-27	Brotto D.	Brotto D.	Approvazione e validazione del documento
v 0.9.0	2024-04-24	Visentin S.	Basso L.	Aggiornata la sezione 4.1, aggiunto il *way of working*$_G$ per ogni ruolo; aggiornato il *versionamento*$_G$ dei documenti
v 0.8.4	2024-04-21	Visentin S.	Basso L.	Modificata sezione 3.2.3 e 3.2.4
v 0.8.3	2024-04-19	Tonietto F.	Basso L.	Aggiunta la sezione 3.5, riguardante la Validazione del prodotto software

v 0.8.2	2024-04-05	Zambon M.	Basso L.	Tolte maiuscole ad alcuni termini
v 0.8.1	2024-04-03	Zambon M.	Basso L.	Cambiate alcuni termini che sono presenti nel glossario per risolvere alcuni problemi con l'automazione dei riferimenti
v 0.8.0	2024-03-28	Zambon M.	Zaupa R.	Modifica della sottosezione 3.1.12 (Glossario tecnico): ridefinita la parte di redazione del glossario e definita la sua verifica; stesura sottosezione 1.1 (scopo del documento) e 1.2 (scopo del prodotto)
v 0.7.2	2024-03-27	Zambon M.	Zaupa R.	Stesura della sottosezione 3.4 (Verifica)
v 0.7.1	2024-03-26	Zambon M.	Zaupa R.	Stesura della sottosezione 3.3 (Gestione della qualità)
v 0.7.0	2024-03-21	Basso L.	Brotto D.	Stesura delle sottosezioni 3.1.2, 3.1.3 (relative ai *Repository*$_G$ per la documentazione) e 3.1.9, 3.1.11, 3.1.12 (relative alle procedure per produrre documentazione)
v 0.6.0	2024-01-18	Basso L.	Tonietto F.	Stesura delle sezione 4 (*processi organizzativi*$_G$) con relative sottosezioni.
v 0.5.0	2024-01-09	Zambon M.	Tonietto F.	Stesura delle sezione 2.4.2 (*Analisi Dei Requisiti*$_G$)
v 0.4.2	2023-12-12	Visentin S.	Tonietto F.	Prima stesura della sezione 2 *Processi primari*$_G$
v 0.4.1	2023-12-09	Visentin S.	Tonietto F.	Piccole modifiche nella sezione 4.
v 0.4.0	2023-11-19	Tonietto F.	Zaupa R.	Stesura della sotto-sottosezione 3.2.4 (relativa al controllo del flusso, con approfondimento sul flusso della documentazione prodotta) e della sottosezione 4.1 (breve analisi e descrizioni dei ruoli di progetto).
v 0.3.0	2023-11-12	Basso L.	Zambon M. Visentin S. Zaupa R.	Stesura della sottosezione 3.2 della sezione relativa alla documentazione e modifiche relative al ciclo di vita di un documento (sottosezione 3.1, parte 3.1.3).
v 0.2.0	2023-11-12	Basso L.	Zaupa R.	Stesura della sezione 3.1 (Documentazione) e delle sottosezioni relative ad essa
v 0.1.0	2023-10-30	Visentin S.	Tonietto F.	Prima versione

Indice

1 Introduzione

1.1 Scopo del documento

Il presente documento si pone lo scopo di identificare le *best practices* di progetto e definire il Way of Working adottato da parte del gruppo, in modo da garantire omogeneità e coesione del lavoro. Per la stesura si utilizza un approccio incrementale e ogni aggiornamento avverrà successivamente a decisioni prese dal gruppo durante l'intera durata del progetto. Ciascun membro del team si impegna a visionare regolarmente tale documento e seguirne le procedure riportate. Eventuali termini tecnici sono definiti all'interno del documento "*Glossario Tecnico*".

1.2 Scopo del prodotto

Il prodotto finale, realizzato tramite un'*Applicazione Web Responsive*$_G$, si propone di realizzare un *software*$_G$ innovativo volto a semplificate il *processo*$_G$ di *prenotazione*$_G$ e *ordinazione*$_G$ nei ristoranti, contribuendo a migliorare l'esperienza per clienti e ristoratori. In particolare, *Easy Meal* dovrà consentire agli utenti di personalizzare gli ordini in base alle proprie preferenze, allergie ed esigenze alimentari; interagire direttamente con lo staff del ristorante attraverso una chat integrata e in ultimo, consentire di dividere il conto tra i partecipanti al tavolo.

1.3 Riferimenti

1.3.1 Riferimenti normativi

1. Presentazione del *capitolato*$_G$ d'appalto C3 - Progetto *Easy Meal*$_G$:
   `https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C3.pdf`
   (consultato: 2024-05-13)

2. Regolamento del progetto didattico:
   `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf`
   (consultato: 2024-05-13)

3. Stardard ISO/IEC 12207:1995:
   `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf`
   (consultato: 2024-05-13)

1.3.2 Riferimenti informativi

1. Lezione *"I processi di ciclo di vita del software*$_G$ *(T2)"* del corso di *ingegneria del software*$_{GG}$:
   `https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T2.pdf`
   (consultato: 2024-05-13)

2. Glossario v2.0.0.

2 Processi primari

Con la locuzione "*processi primari*$_G$" si intendono "tutti i processi che hanno come clienti soggetti esterni al gruppo".

2.1 Acquisizione

Nel *processo*$_G$ di acquisizione avviene la raccolta e la comprensione dei requisiti con lo scopo di identificare un *capitolato*$_G$ adeguato per il gruppo da proporre per la *candidatura*$_G$.

2.1.1 Valutazione capitolati

Il gruppo ha analizzato la proposta dei proponenti valutandone, in base all'esperienza del gruppo, la loro *complessità*; tale analisi è stata determinante per la scelta definitiva.

2.1.2 Appalto capitolati

Il gruppo si è proposto per il *capitolato*$_G$ C3 dell'azienda *Imola Informatica*$_G$. Nonostante un un primo riscontro negativo, a seguito di modifiche volte a sistemare lacune presenti presenti nella *candidatura*$_G$ iniziale, il gruppo è riuscito ad aggiudicarsi il *capitolato*$_G$.

2.2 Fornitura

Il *processo*$_G$ di *fornitura*$_G$ consiste nel chiarire ogni dubbio legato al prodotto finale che il proponente desidera; in modo da evitare incomprensioni durante lo svolgimento del progetto, il gruppo RAMtastic6 si impegna a comunicare con l'azienda, in modo da raggiungere i seguenti obiettivi:

1. Determinare i requisiti da soddisfare nel prodotto finale;
2. Ottenere incontri di formazioni sulle *tecnologie*$_G$ e strumenti consigliati dall'azienda per realizzare il prodotto;
3. Ricevere *feedback*$_G$ in fase di sviluppo su quanto precedentemente svolto.

2.2.1 Lettera di Presentazione

La *Lettera di Presentazione* è il documento che fornisce la presentazione della documentazione e del prodotto durante le varie fasi di revisione di progetto.
Questo documento riporta l'insieme della documentazione richiesta che sarà consegnata ai committenti, il Prof. Tullio Vardanega e il Prof. Riccardo Cardin.

2.2.2 Analisi dei Requisiti v3.0.0

L'*Analisi dei Requisiti* stabilisce una comprensione comune tra gli *stakeholder* riguardo alle caratteristiche e alle prestazioni del *software*$_G$ da sviluppare. Questo documento elimina le ambiguità che potrebbero sorgere durante la comunicazione e fornisce una base chiara per il progetto.
Il documento di *Analisi dei Requisiti* è strutturato in diverse sezioni chiave:

- **Introduzione**: Fornisce una panoramica generale del prodotto e delle sue funzionalità principali. Questa sezione stabilisce il contesto per il resto del documento e fornisce una visione d'insieme del progetto.

- **Casi d'uso**: Identifica e descrive tutti i possibili scenari di utilizzo del *software*$_G$ da parte degli utenti. Ogni *caso d'uso*$_G$ descrive un'interazione specifica tra l'utente e il *sistema*$_G$, illustrando come il *software*$_G$ dovrebbe essere utilizzato per soddisfare le esigenze degli utenti.

- **Requisiti**: Raccoglie tutte le richieste e i vincoli definiti dal cliente o emersi durante le discussioni con il team di sviluppo. Ogni *requisito*$_G$ è accompagnato da una descrizione dettagliata che specifica ciò che il *sistema*$_G$ deve fare o quale comportamento deve avere. Questi requisiti costituiscono la base per la progettazione e lo sviluppo del *software*$_G$.

2.2.3 Specifica Tecnica v1.0.0

Il documento *Specifica Tecnica* descrive in modo dettagliato le scelte progettuali effettuate dal gruppo per la realizzazione del *sistema*$_G$ richiesto dal proponente.
Viene compresa l'architettura logica e l'architettura di *deployment* oltre che la lista delle *tecnologie*$_G$ utilizzate e i *design*$_G$ pattern adottati. Inoltre, viene fornita una sezione relativa al tracciamento dei requisiti soddisfatti in linea con il documento di *Analisi dei Requisiti*.

2.2.4 Manuale Utente v1.0.0

Il *Manuale Utente* ha lo scopo di fornire all'utilizzatore del prodotto un orientamento verso quelli che sono tutti gli scenari in cui potrà navigare l'utente una volta messo in atto.
Comprende una guida di navigazione completa per eliminare tutti i dubbi che potrebbe avere durante l'utilizzo di tutte le funzionalità che il prodotto ha da offrire.

2.2.5 Glossario v2.0.0

Il *Glossario* è il documento alla base di ogni altro documento di supporto. Il suo scopo è quello di esplicitare il significato dei termini comunemente usati all'interno di tutta la documentazione prodotta, al fine di evitare ambiguità o incomprensioni.
E' rivolto sia ai componenti del gruppo che ai committenti e ai proponenti.

2.2.6 Piano di Progetto v2.0.0

Il documento *Piano di Progetto* costituisce uno *strumento*$_G$ di pianificazione per tutte le attività che dovranno essere svolte così da poter rispettare la data di consegna del progetto. Nel dettaglio il *Piano di Progetto*$_G$ è composto da:

- *Analisi Dei Rischi*$_G$: analisi delle difficoltà che il gruppo potrebbe riscontrare durante lo svolgimento del progetto, in particolar modo a livello organizzativo e tecnologico;

- Modello di sviluppo;

- Pianificazione;

- *Preventivo*$_G$: che rispecchi quanto comunicato in fase di *candidatura*$_G$;

- *Consuntivo*$_G$: tracciamento dell'andamento del gruppo rispetto al *preventivo*$_G$ fatto.

2.2.7 Piano di Qualifica v2.0.0

Nel documento *Piano di Qualifica* vengono elencate tutte le attività svolte dal verificatore con l'obiettivo di garantire la qualità del prodotto finale. E' formato dai seguenti componenti:

- Qualità di processo
- Qualità di prodotto
- $Test_G$: eseguiti sul prodotto che assicurano che i requisiti siano stati rispettati
- Resoconto

2.2.8 Rilascio

Quando il prodotto verrà ultimato, verrà collaudato per garantire il suo corretto funzionamento. Se il prodotto supera il collaudo, verrà consegnato al committente insieme alla documentazione del progetto. Il gruppo non effettuerà $manutenzione_G$ una volta rilasciato il prodotto.

2.2.9 Strumenti

Per il $processo_G$ di $fornitura_G$ vengono utilizzati i seguenti strumenti:

- ***Microsoft Teams***: servizio di videoconferenza utilizzato per gli incontri con il proponente;
- ***Telegram***: servizio di messaggistica utilizzato per le comunicazioni con il proponente;
- ***Share Point***: servizio che permette di condividere e gestire contenuti Office tramite browser, utilizzato per creare le presentazioni per i diari di bordo.

2.3 Sviluppo

Lo scopo del $processo_G$ di sviluppo è quello di dichiarare le attività da svolgere per raggiungere i requisiti necessari del prodotto. A tale scopo, il gruppo si dividerà in diversi ruoli, i quali avranno compiti precisi da svolgere.

2.3.1 Analisi Dei Requisiti

Analisi Dei Requisiti è un documento fondamentale per lo sviluppo. Infatti, tale documento deve indicare i requisiti necessari del prodotto finale, che a loro volta andranno a rispecchiare le aspettative del proponente.
Inoltre, questo documento servirà anche come documentazione del prodotto, andandone infatti a contenere tutte le relative Funzionalità.

2.3.1.1 Attori

Innanzitutto, verranno definiti gli attori e una panoramica dei vari casi d'uso a loro associati tramite un diagramma

2.3.1.2 Casi d'uso

Successivamente, verranno descritti i vari casi d'uso, i quali hanno il compito di rappresentare le Funzionalità che il prodotto finale dovrà rispettare.
I casi d'uso saranno ordinati per *attore*$_G$, ovvero verranno prima descritti tutti i casi d'uso associati a un particolare *attore*$_G$, per poi proseguire con la descrizione di tutti i casi d'uso riguardanti l'*attore*$_G$ successivo, e a proseguire.
Ogni Caso d'Uso avrà un diagramma ad esso associato e la sua descrizione *verbale*$_G$. Gli use case cosiddetti "atomici" saranno inseriti nello *scenario*$_G$ principale, il quale rappresenterà le azioni compiute in modo consecutivo dall'*attore*$_G$ nello *scenario*$_G$ più comune.
La descrizione *verbale*$_G$ di ogni *caso d'uso*$_G$ seguirà la seguente struttura standard:

- Attori
- Precondizioni
- Postcondizioni
- *Scenario*$_G$ primario
- Scenari alternativi (opzionale)

Gli scenari alternativi saranno presenti solo se il Caso d'Uso può generare eccezioni: poiché a più eccezioni corrispondono una singola modalità di gestione delle stesse, per ogni *scenario*$_G$ alternativo ci potranno essere più primi punti, i quali verranno rappresentati nel formato "*1.xa*", nel quale '*x*' rappresenta il punto dello *scenario*$_G$ principale dal quale viene generata l'*eccezione*$_G$, mentre '*a*' rappresenta il tipo di *eccezione*$_G$. Verrà poi descritto dal punto "*2*" a seguire lo *scenario*$_G$ alternativo.

2.3.1.3 Requisiti

Verranno infine descritti i requisiti, che potranno essere di tre tipi:

- **Funzionali**: requisiti che delineano le varie funzionalità del *sistema*$_G$, ovvero le azioni eseguibili dallo stesso e le informazioni che il *sistema*$_G$ è in grado di fornire;
- **Di qualità**: requisiti che delineano le specifiche qualitative che devono essere rispettate al fine di garantire la qualità del *sistema*$_G$;
- **Requisiti di vincolo**: requisiti che delineano i limiti e le restrizioni di cui il *sistema*$_G$ deve tener conto per adempiere alle esigenze del proponente.

L'elenco di tutti i tipi di requisiti presenterà la seguente struttura:

- *Codice*: un codice identificativo univoco per ogni *requisito*$_G$;
- *Descrizione*: una descrizione *verbale*$_G$ del *requisito*$_G$;
- *Fonte*: l'elemento che ha generato la necessità di produrre il *requisito*$_G$ preso in esame; può essere uno o più casi d'uso, come pure direttamente il testo del *capitolato*$_G$ d'appalto.

2.3.1.4 Strumenti

- ***LucidChart***: applicazione *software*$_G$ utilizzata dal team per la realizzazione dei diagrammi dei casi d'uso.

2.3.2 Progettazione

La progettazione, a carico della figura del progettista, definisce la struttura del progetto basandosi sull'*Analisi Dei Requisiti*. La progettazione avviene su più livelli:

1. *Progettazione architetturale*$_G$: dove viene scelta la struttura del sistema
2. *Design*$_G$: ovvero il design dell'interfaccia che deve avere il prodotto
3. *Progettazione dettagliata*$_G$: le specifiche dei componenti del prodotto che comprendono le specifiche architetturali, i diagrammi delle classi e i *test*$_G$ d'unità

2.3.2.1 Obiettivi

Nella fase di progettazione di un prodotto *software*$_G$, l'obiettivo principale è garantire che i requisiti siano soddisfatti attraverso un *sistema*$_G$ di qualità definito dall'*architettura*$_G$ del prodotto.
È essenziale organizzare il *sistema*$_G$ in modo da facilitare futuri adattamenti e gestire la complessità mediante una *progettazione dettagliata*$_G$ che suddivide il *sistema*$_G$ in unità architetturali, rendendo più semplice la *codifica*$_G$ di ogni parte e assicurando che sia gestibile, veloce e verificabile.
Il team di progettazione inizia con un'analisi approfondita per selezionare le *tecnologie*$_G$ più appropriate, valutandone i vantaggi, le debolezze e le potenziali criticità.
Una volta scelte le *tecnologie*$_G$, si sviluppa un'*architettura*$_G$ di alto livello per delineare la struttura generale del prodotto, creando una base solida per la realizzazione di un *Proof of Concept* (PoC_G).
Questa *architettura*$_G$ fornisce una visione d'insieme del *sistema*$_G$, identificando i principali componenti, i flussi di dati e le loro interazioni, con un'attenzione particolare alla flessibilità per future modifiche.
Il PoC_G serve a valutare le decisioni architetturali e tecnologiche e a verificarne la conformità agli obiettivi e alle specifiche del progetto. Dopo lo sviluppo e l'analisi del PoC_G, si procede con iterazioni successive per migliorare, aggiustare e completare il *design*$_G$, fino a ottenere un *Minimum Vaulable Product* (MVP_G), che rappresenta una versione funzionale ed essenziale del prodotto, integrata nella *Product Baseline*.

2.3.2.2 Documentazione

Specifica Tecnica
Il documento di *Specifica Tecnica* descrive dettagliatamente il *design*$_G$ finale del prodotto e fornisce istruzioni precise agli sviluppatori per guidarli nella corretta implementazione della soluzione *software*$_G$ in linea con i requisiti e le specifiche indicate. Questo documento è fondamentale per ridurre la complessità e le ambiguità nel *processo*$_G$ di sviluppo, assicurando che il prodotto finale soddisfi le aspettative del cliente e funzioni in modo ottimale.
Il documento di *Specifica Tecnica* include vari elementi essenziali:

- *Architettura*$_G$ logica: definisce i componenti, i ruoli, le connessioni e le interazioni all'interno del *sistema*$_G$.

- *Architettura*$_G$ di *deployment*: descrive come i componenti architetturali vengono allocati e distribuiti nel *sistema*$_G$ in esecuzione.
- *Design*$_G$ pattern: spiega i design pattern architetturali adottati e quelli influenzati dalle *tecnologie*$_G$ utilizzate.
- Procedure di *testing*: indica i processi per testare e verificare che il *software*$_G$ soddisfi i requisiti specificati.
- Requisiti tecnici: dettaglia i requisiti prestazionali, di sicurezza, di scalabilità e di compatibilità con determinate piattaforme che il *software*$_G$ deve soddisfare.

Attraverso una documentazione accurata e completa, la *Specifica Tecnica* funge da riferimento chiave per il team di sviluppo, facilitando una comprensione chiara e condivisa del progetto e contribuendo a un prodotto finale di alta qualità.

2.3.2.3 Qualità dell'architettura

La qualità dell'*architettura*$_G$ di un *software*$_G$ è fondamentale per assicurare che il *sistema*$_G$ non solo soddisfi i requisiti funzionali e non funzionali, ma che lo faccia in maniera efficiente e sostenibile. Gli aspetti chiave da considerare includono:

- **Adeguatezza**: l'*architettura*$_G$ deve rispondere ai requisiti funzionali e non funzionali definiti, evitando sia sovra-dimensionamenti che sotto-dimensionamenti.
- **Chiarezza**: l'*architettura*$_G$ deve essere facilmente comprensibile, permettendo a sviluppatori e *stakeholder* di afferrare rapidamente il funzionamento del *sistema*$_G$.
- **Modularità**: l'*architettura*$_G$ deve essere suddivisa in moduli o componenti ben definiti, facilitando la separazione delle responsabilità, la *manutenzione*$_G$, l'aggiornamento e lo sviluppo parallelo.
- **Disponibilità**: il *sistema*$_G$ deve essere accessibile e operativo quando richiesto, minimizzando i tempi di inattività non pianificati.
- **Flessibilità**: l'*architettura*$_G$ deve permettere adattamenti o estensioni per nuovi requisiti o cambiamenti, senza necessitare di modifiche radicali.
- **Semplicità**: l'*architettura*$_G$ deve essere il più semplice possibile, senza compromettere funzionalità o efficacia.
- **Efficienza**: il *sistema*$_G$ deve utilizzare in modo ottimale le risorse disponibili, come memoria e CPU, eseguendo le sue funzioni nel minor tempo possibile.
- **Basso accoppiamento**: le interazioni o dipendenze tra i diversi moduli o componenti devono essere minimizzate per migliorare la manutenibilità e la flessibilità del *sistema*$_G$.
- **Robustezza**: il *sistema*$_G$ deve essere in grado di gestire situazioni anomale o errori senza causare gravi interruzioni o perdite di dati, mantenendo un funzionamento accettabile.
- **Sicurezza operativa (Safety)**: deve prevenire danni fisici o danni a persone e beni causati da malfunzionamenti del *software*$_G$.

- **Sicurezza contro intrusioni (Security)**: il *software*$_G$ deve essere protetto da accessi non autorizzati, manipolazioni e intrusioni esterne.
- **Riusabilità**: i componenti del *software*$_G$ devono poter essere riutilizzati in contesti diversi, riducendo il carico di sviluppo e migliorando l'efficienza.
- **Incapsulamento**: i dettagli implementativi devono essere nascosti all'esterno di un componente, accessibili solo attraverso un'interfaccia definita.
- **Coesione**: i componenti all'interno di un modulo devono lavorare insieme per un obiettivo comune, evitando eccessive interdipendenze.
- **Affidabilità**: il *software*$_G$ deve funzionare correttamente e in modo coerente nel tempo, garantendo una prestazione prevedibile.

Questi aspetti sono cruciali per creare un'*architettura*$_G$ *software*$_G$ robusta, flessibile e manutenibile, in grado di soddisfare le esigenze presenti e future del progetto.

2.3.2.4 Diagrammi delle classi

I diagrammi delle classi rappresentano le proprietà e le relazioni dei componenti gli uni con gli altri. Dei rettangoli rappresentano graficamente le classi, mentre invece, diversi tipi di frecce rappresentano la relazioni tra le classi. I diversi tipi di relazioni che si incontrano sono:

- **Dipendenza**
  La relazione di dipendenza tra due classi, A e B, indica che la classe A dipende dalla classe B per la sua specifica, implementazione o funzionamento. Questo tipo di relazione è rappresentata da una freccia tratteggiata che punta da A (il cliente) verso B (il fornitore). Ad esempio, se la classe A utilizza i metodi o gli attributi della classe B, un cambiamento nella classe B potrebbe influire sul corretto funzionamento della classe A.

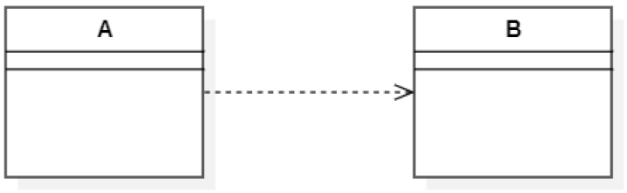

Figura 1: Diagramma UML della relazione Dipendenza

- **Composizione**
  La relazione di composizione tra due classi, A e B, implica che l'oggetto della classe A è composto da oggetti della classe B, e che l'esistenza degli oggetti della classe B dipende direttamente dall'esistenza degli oggetti della classe A. Questa relazione è rappresentata da una freccia solida con un rombo pieno alla fine, che punta dalla classe A alla classe B.

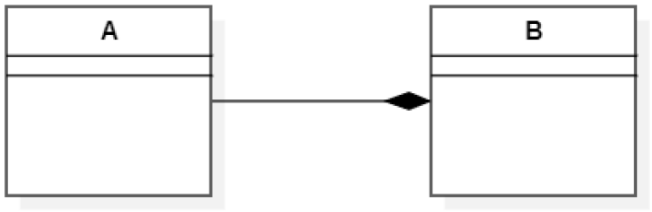

Figura 2: Diagramma UML della relazione Composizione

- **Aggregazione**
  La relazione di aggregazione tra due classi, A e B, implica che l'oggetto della classe A "aggrega" gli oggetti della classe B, ma questi ultimi possono esistere anche al di fuori del contesto della classe A. È rappresentata da una freccia con un rombo vuoto alla fine, che punta dalla classe A alla classe B.

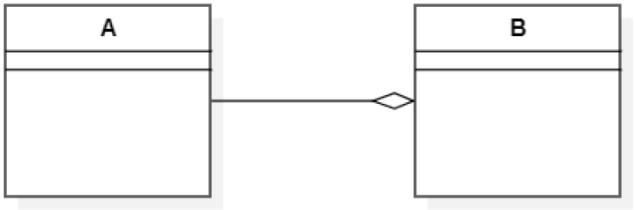

Figura 3: Diagramma UML della relazione Aggregazione

- **Relazione con interfaccia**
  La relazione con interfaccia tra due classi, A e B, è indicata da una linea non direzionata che collega la classe A a un cerchio vuoto che rappresenta l'interfaccia fornita dalla classe B. Questa relazione denota che la classe A dipende dall'interfaccia definita dalla classe B per specificare le sue operazioni, senza specificare l'implementazione concreta di tali operazioni.

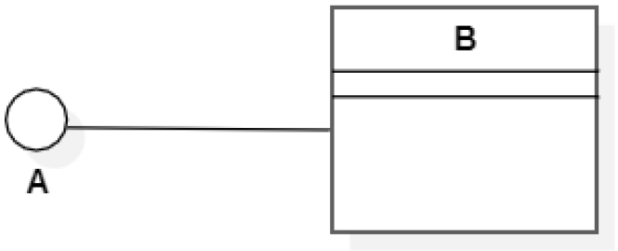

Figura 4: Diagramma UML della relazione Interfaccia

2.3.2.5 Design Pattern

I *design patterns* sono soluzioni consolidate a problemi di progettazione che si presentano in modo ricorrente in diversi contesti.
Questi modelli offrono un approccio riusabile alla progettazione, garantendo qualità nella soluzione e velocità nell'implementazione.
L'adozione di un *design pattern* avviene quando una soluzione si è dimostrata efficace in un contesto specifico, fornendo una guida affidabile per affrontare problemi simili in futuro.
Di solito, vengono fornite dettagliate linee guida sull'applicazione dei pattern, insieme a una rappresentazione grafica e una spiegazione testuale della loro logica e utilità all'interno dell'*architettura*$_G$ complessiva.
Questa documentazione gioca un ruolo essenziale nel favorire la comprensione dell'integrazione del *design*$_G$ pattern nell'*architettura*$_G$ generale e nella prevenzione di errori di progettazione, assicurando coerenza e coesione nel *sistema*$_G$ *software*$_G$.

2.3.2.6 Metriche

Metrica	Nome
MPD09-BS	`Browser supportati`
MPD04-TA	`Tempo di apprendimento`
MPD05-RO	`Raggiunta dell'obbiettivo`
MPD06-EU	`Errori dell'utente`
MPD03-TM	`Tempo di risposta medio`

2.3.2.7 Strumenti

- ***LucidChart***: applicazione *software*$_G$ utilizzata dal team per la realizzazione dei diagrammi riguardanti l'*architettura*$_G$ logica.
- ***StarUML***: applicazione *software*$_G$ utilizzata dal team per la realizzazione dei diagrammi delle classi.

2.3.3 Codifica

I programmatori, dopo l'analisi e la progettazione, implementano le funzionalità che deve avere il prodotto finale basandosi sull'*Analisi Dei Requisiti* e sui documenti di progettazione.
Le aspettative riguardanti tale attività nel caso della *codifica*$_G$ dell'*MVP*$_G$ sono le seguenti:

- Le modifiche attuate dal programmatore devono essere coerenti con i requisiti e i casi d'uso presenti nel documento *Analisi dei Requisiti*. Nel caso in cui fosse necessario rivedere alcuni sezioni si discute con l'analista su come procedere.
- Le modifiche attuate dal programmatore devono essere coerenti con l'*architettura*$_G$ e la progettazione del documento *Specifica Tecnica*. Nel caso in cui fosse necessario rivedere alcuni sezioni si discute con il progettista su come procedere.
- Rispetto a quanto implementato si devono inserire *test*$_G$ di unità che assicurano il corretto funzionamento del prodotto.

2.3.3.1 Norme di codifica

Per la stesura del codice si sono adottate delle norme comuni tra cui:

- **Convenzioni di denominazione**: Usare nomi significativi per variabili, funzioni, classi, ecc. usando opportunamente lettere maiuscole e minuscole per identificare le parole chiave.
- **Indentazione**: Usa l'indentazione in modo consistente per evidenziare la struttura del codice.
- **Commenti**: Aggiungere commenti significativi per spiegare parti complesse del codice, ma evitare commenti ovvi o ridondanti.
- **Lunghezza delle righe**: Limitare la lunghezza delle righe per una facile lettura e revisione del codice.
- **Struttura del codice**: Organizzare il codice in blocchi logici e funzioni coese. Utilizzare funzioni e classi per evitare la duplicazione del codice.
- **Gestione delle eccezioni**: Trattare le eccezioni in modo appropriato, usando blocchi try-catch quando necessario. Non ignorare mai le eccezioni senza gestirle.
- **Separare *logica di business*$_G$ e logica di interfaccia**: Mantenere separata la *logica di business*$_G$ dal codice di interfaccia utente o di accesso ai dati. Questo rende il codice più modulare e facile da testare.

2.3.3.2 Metriche

Metrica	Nome
MPD08-CC	`Complessità ciclomatica`
MPD07-FD	`Failure density`
MPC15-PCTS	`Percentuale dei` *test*$_G$ `superati`
MPC16-SC	`Statement coverage`
MPC17-BC	`Branch coverage`
MPC18-CC	`Condition coverage`
MPC12-LOC	`Linee di codice`

2.3.3.3 Strumenti

- ***Visual Studio Code***: IDE utilizzato dal gruppo per la *codifica*$_G$ del prodotto *software*$_G$.

3 Processi di supporto

3.1 Documentazione

3.1.1 Tipologie di documenti

I documenti prodotti possono essere classificati in due classi principali: ad uso interno e ad uso esterno; la prima categoria comprende:

- *Verbali interni*$_G$ (i quali non necessitano di *versionamento*$_G$);
- *Norme di Progetto*$_G$.

La seconda categoria di documenti comprende:

- *Verbali esterni*$_G$;
- *Piano di Qualifica*$_G$;
- *Piano di Progetto*$_G$;
- *Analisi Dei Requisiti*$_G$;
- Glossario Tecnico;
- Specifica Tecnica.

3.1.2 Gestione della documentazione

Una particolare attenzione in tal senso è rivolta alla documentazione. Al fine di mantenere nel *repository*$_G$ solamente i *PDF*$_G$ dei documenti prodotti, è stato deciso di adottare la piattaforma *Overleaf*$_G$ per la stesura in *LaTeX*$_G$ dei documenti e la successiva verifica. Ogni volta che un documento viene redatto o aggiornato, verificato e portato alla versione corretta come precedentemente indicato, può essere comodamente convertito in formato *PDF*$_G$ tramite *Overleaf*$_G$. Successivamente, il documento può essere caricato nella *repository*$_G$, con il *push*$_G$ diretto sul *branch*$_G$ *develop*, soprattutto quando si parla di documentazione importante e la cui stesura è in itinere.

3.1.3 Struttura di un progetto in Overleaf

Affinché le automazioni per produrre *riferimenti*$_G$ al glossario funzionino, un progetto in *Overleaf*$_G$ deve avere la struttura seguente:

- main.tex : file contenente il contenuto principale o, se si vuole, tutto il contenuto del file.
- CONTENTS/ : cartella contenente ulteriori file (.tex) che il main.tex usa per poter costruire un documento unico

Per favorire una migliore collaborazione si è deciso di creare un file sorgente per ciascuna sezione del documento presente nella cartella CONTENTS/. Tale approccio viene utilizzato per i documenti di *Norme di Progetto*$_G$, *Piano di Qualifica*$_G$, *Piano di Progetto*$_G$, Analisi dei Requisiti e Glossario Tecnico. Per i verbali il progetto viene riportato interamente nel file main.tex.

3.1.4 Struttura di un documento

Un documento all'interno del nostro contesto segue una struttura ben definita, le sue sezioni principali includono:

- Prima pagina: contiene il nome del gruppo e informazioni in merito al documento: uso, destinatari, redattori, verificatori, versione
- Indice: elenco strutturato dei contenuti del documento
- Registro dei cambiamenti: una tabella contente informazioni di *versionamento*$_G$ relative al documento attuale; queste includono: la versione, la data, l'autore, il verificatore e una breve descrizione in merito alle modifiche apportate al documento.
- Intestazione: all'interno di essa vi sono il nome e l'indirizzo email del gruppo.

3.1.5 Ciclo di vita di un documento

Un documento segue le seguenti fasi di produzione:

- Stesura: uno o più redattori si occupano di redigere il contenuto del documento.
- Verifica: ad uno o più membri del gruppo, diversi da quelli che hanno redatto il documento, viene assegnato il compito di verifica del documento. È importante sottolineare che tutti i documenti sopracitati sono ufficiali e devono essere, quindi, preventivamente approvati da verificatori designati.
- Approvazione: durante questa fase, il responsabile di progetto può decidere se approvare l'inclusione di un particolare documento all'interno del *repository*$_G$. Nel caso in cui il documento non venga approvato, si ritorna alla fase di stesura. Se quest'ultima fase va a buon fine, vengono aggiunte informazioni di *versionamento*$_G$ secondo quanto riportato nell'apposita sezione; infine viene caricato il documento all'interno del *repository*$_G$ nel *branch*$_G$ develop.

3.1.6 Procedure branch develop

Il flusso di lavoro attuale per produrre la documentazione relativa al progetto è il seguente:

1. modificare i documenti in Overleaf
2. scaricare i sorgenti dei documenti scritti in *Overleaf*$_G$ e inserirli all'interno del *repository*$_G$ locale Project14, nella cartella sorgenti.
3. posizionarsi nella cartella Project 14 ed eseguire le automazioni mediante il comando:

   python3 entry_point_automazioni.py

   Il risultato di questa esecuzione produrrà dei nuovi sorgenti (.tex): in particolare:

   (a) Una versione aggiornata del glossario tecnico
   (b) Gli stessi file che contenevano *riferimenti*$_G$ al glossario tecnico dove ve ne siano.

3.1.7 Glossario Tecnico

Il *Glossario tecnico* è un documento di supporto concepito per evitare ambiguità o incomprensioni riguardanti la terminologia utilizzata in tutta la documentazione per ogni fase del progetto ed è rivolto sia ai componenti del gruppo che a committenti e proponenti. Si tratta dell'unico documento da non modificare all'interno di $Overleaf_G$. Infatti, il glossario tecnico viene costruito a partire da un file in formato *json* contenuto all'interno della cartella *sorgenti/Glossario* del $repository_G$ *Project14*. Tale file, denominato *glossario.json*, è costituito da un array di oggetti; ogni oggetto è formato da un insieme di due coppie chiave-valore, in particolare vi sono:

- La chiave termine: il termine che necessita di essere definito all'interno del dominio di progetto;
- Il valore definizione: la definizione del termine stesso.

Redazione Per poter inserire o modificare un termine nel glossario tecnico, bisogna seguire i seguenti passaggi:

- Creare un $branch_G$ nel $repository_G$ *Project14* associato al $ticket_G$ che indica le parole da inserire o modificare all'interno del glossario;
- Aggiungere o modificare le parole e definizioni nel file *glossario.json*;
- modificare il $changelog_G$ nel file *build_glossary.py* nella cartella GLOSSARY_AUTOMATIONS;
- Eseguire l'automazione tramite il comando:

  python3 entry_point_automazioni.py

- Effettuare il $push_G$ in remoto;
- Aprire una $pull\ request_G$ dal $branch_G$ creato al $branch_G$ develop;
- Spostare il $ticket_G$ corrispondente nella sezione "da verificare" all'interno di *Jira*.

Nel momento in cui viene inserita una parola nuova all'interno del glossario bisogna segnalare al responsabile eventuali discrepanze tra il modo in cui è stato scritto il termine all'interno del file *glossario.json* e il modo in cui è stato scritto all'interno dei documenti. Viene riportato il seguente esempio: se all'interno del glossario viene riportata la parola "*Analisi Dei Requisiti*" allora all'interno dei documenti tale parola deve essere riportata con le stesse lettere maiuscole e minuscole eccetto la lettera iniziale (va bene "*analisi Dei Requisiti*" ma non "*Analisi dei Requisiti*").

Verifica Per poter verificare il glossario, si seguono le seguenti azioni:

- Utilizzare il $branch_G$ creato dal $redattore_G$;
- Verificare i termini e definizioni nel file *glossario.json*;
- Modificare il $changelog_G$ nel file *build_glossary.py* nella cartella GLOSSARY_AUTOMATIONS;
- Eseguire l'automazione tramite il comando:

  python3 entry_point_automazioni.py

- Effettuare il *push*$_G$ in remoto;
- Spostare il *ticket*$_G$ nella sezione "verificato" all'interno di *Jira*.

Infine, il responsabile approva la *pull request*$_G$ associata.

3.1.8 Strumenti

- ***Overleaf***$_G$, un *editor*$_G$ *LaTeX*$_G$ online che permette la stesura collaborativa dei documenti. Inoltre, il servizio rende disponibili dei *template*$_G$ che il gruppo ha scelto di utilizzare per la produzione della documentazione. Per visualizzare la struttura e utilizzare i *template*$_G$, è sufficiente cercarli su *Overleaf*$_G$.

3.2 Controllo di Configurazione

3.2.1 Versionamento

Il *versionamento*$_G$ scelto per tenere traccia dei documenti è una tripletta di numeri: x.y.z.

- x è un numero intero, il quale, partendo da 0, verrà incrementato ogni volta che il responsabile approva tale documento;
- y è un numero intero positivo, e rappresenta lo stato di verifica del documento;
- z è un numero intero positivo, e rappresenta il singolo cambiamento apportato al file.

3.2.2 Git e Github

Il gruppo RAMtastic6 ha scelto di utilizzare come *strumento*$_G$ di *versionamento*$_G$ *Github*$_G$ e di utilizzare *Git* come *strumento*$_G$ per collegarsi alla *repository*$_G$ *Github*$_G$. Inoltre si è scelto di utilizzare *Gitflow*$_G$ come flusso di lavoro il quale verrà discusso in modo dettagliato in seguito (Link per il download dell'installer di Git).
Inoltre, a questo link si troverà una breve guida su come utilizzare *Git*$_G$. In sintesi si elencano i principali comandi:

- *git*$_G$ clone *link repo*
  questo comando copierà la *repository*$_G$ di *Github*$_G$ in locale
- *git*$_G$ add *nome file* (oppure "." per includere tutti i file)
  git add aggiunge le modifice apportate ai files del *repository*$_G$, senza eseguire questo comando un file aggiunto, eliminato o modificato non verrà salvato nella *repository*$_G$ remota tramite il comando *git push*.
- *git*$_G$ commit -m "messaggio"
  salva le modifche apportate ai files in locale associando a quello stato un messaggio
- *git*$_G$ *push*$_G$ origin *origine*
  salva le modifiche in remoto nel *branch*$_G$ specificato
- *git*$_G$ pull
  permette di aggiornare la repo in locale e in caso di necessità esegue il merge

3.2.3 Repository

Il gruppo utilizza i seguenti *repository*$_G$ all'interno della propria organizzazione GitHub:

- ***Project14***: contenente la documentazione del progetto;
- ***Proof-of-Concept***: dedicato al *PoC*$_G$ del progetto.

Si segnala che è presente inoltre il *repository*$_G$ GitHub contenete il sito web del gruppo, non rilevante ai fini del documento.

3.2.3.1 Scopo e struttura repository "Project14"

Questo *repository*$_G$ ha due funzioni:

- mantenere una versione aggiornata dei sorgenti atti a produrre documentazione (file .tex)
- disporre di automazioni per produrre automaticamente documentazione (file in formato .pdf) a partire dai sorgenti.

La strutta di questo *repository*$_G$ (link) deve essere:

- documenti
  - Candidatura
  - RTB
  - PB
- diari_di_bordo
- documenti_interni

Un *verbale*$_G$ deve essere nominato utilizzando il seguente formato: *verbale*$_G$_AAAA_MM_GG.

3.2.3.2 Scopo e struttura repository "Proof-of-Concept"

Questo *repository*$_G$ (link) è dedicato al contenimento della *Poc* del progetto.
Presenta la seguente struttura:

- nestjs: contiene il progetto relativo a NestJS e quindi al *backend*$_G$ della web app
- nextjs: contiene il progetto di NextJS, ovvero il frontend
- socket: contiene il progetto che implementa i socket, che serve per implementare l'*ordinazione*$_G$ collaborativa
- postgres: contiene il database della web app

3.2.3.3 Scopo e struttura repository "EasyMeal"

Questo *repository*$_G$ (link) è dedicato al contenimento del codice sorgente dell'intero progetto. Presenta la seguente struttura:

- *.github*$_G$/workflows: contiene tutto l'occorrente per far girare i file di *test*$_G$ direttamente da *github*$_G$;
- nest-js: contiene tutto il progetto relativo a NestJS e quindi al *backend*$_G$ della web app;
- next-js: contiene tutto il progetto di NextJS, ovvero il *frontend*$_G$;
- socket: contiene tutto il progetto che implementa i socket, che serve per implementare l'ordinazione collaborativa e il *sistema*$_G$ di notifiche;
- postgres: contiene il database della web app.

Oltre a tutto ciò, nel file README.md è riportato il link a Coveralls, che riporta la percentuale della copertura del codice dell'intero progetto (dato rilevante per il proponente che ha richiesto una copertura minima dell'80%).

3.2.4 Controllo di Flusso

Il gruppo RAMtastic6 ha deciso di dotarsi di *Gitflow*$_G$ come *sistema*$_G$ di controllo del flusso di lavoro, motivato dalla sua facilità d'uso e dalle potenzialità di gestione offerte per il *repository*$_G$. Con una lieve modifica nei comandi per l'esecuzione dei commit, come illustrato in questa guida su *Gitflow*$_G$, è possibile automatizzare il *processo*$_G$ di creazione, gestione e chiusura di una *feature*$_G$. Ulteriori dettagli su come gestire le *feature*$_G$ sono disponibili a questo link.

3.2.4.1 Gestione dei branch e tickets per la codifica

Seguendo la convenzione GitFlow i *branch*$_G$ sono distinti come segue:

- **Main**: è il *branch*$_G$ di default che contiene la documentazione e il codice che vengono consegnati in fase di revisione. Tale ramo viene dunque aggiornato solo in occasione di tale evento.
- **Develop**: creato a partire dal main, contiene le nuove funzionalità rispetto a tale *branch*$_G$. Viene aggiornato solo in occasione di *rilascio*$_G$ di nuove *feature*$_G$ dagli apposti *branch*$_G$ *feature*$_G$.
- **Feature**: creati a partire da develop, contengono una funzionalità ciasuno sviluppata rispetto a tale *branch*$_G$. Sono i *branch*$_G$ dove avviene la *codifica*$_G$ vera e propria.

Codifica Quando si viene assegnati, tramite i *ticket*$_G$ di *jira*$_G$, a svolgere un'attività riguardante la scrittura di codice, per esempio implementare nuove funzionalità, correlabile ad un *branch*$_G$ *feature*$_G$, allora la procedura da seguire è la seguente:

1. Tramite il *ticket*$_G$ bisogna creare un nuovo *branch*$_G$ (aprire il *ticket*$_G$, nella sezione dettagli e selezionare "Crea *branch*$_G$") il quale dovrà avere il numero del *ticket*$_G$ come inizio del nome per poi scrivere in modo conciso lo scopo del *branch*$_G$ (per esempio: P1-100-prenotazione). In questo modo il nuovo ramo sarà collegato al *ticket*$_G$. Tale *branch*$_G$ dovrà essere generato a partire dal *branch*$_G$ develop;

2. Viene implementata la funzionalità correlata;
3. Il programmatore crea una *pull request*$_G$ su GitHub con l'intenzione di effettuare il merge nel ramo develop; nella *pull request*$_G$ (come viene fatto per il *changelog*$_G$ dei documenti) devono essere riportate le modifiche effettuate in modo da aiutare l'attività di verifica;
4. Nella *pull request*$_G$ si inserisce nella sezione "reviewers" il verificatore incaricato di verificare tale funzionalità.
5. Il verificatore può:
    - Accettare i cambiamenti e accettare la *pull request*$_G$ tramite l'apposito comando;
    - Richiedere ulteriori cambiamenti, in tal caso redige un *feedback*$_G$ nella *pull request*$_G$ e il programmatore provederà a sistemare. Si ripete dal passo 4 fino ad arrivare ad uno stato di accettazione da parte del verificatore.

Come riportato nella sezione riguardante l'attività di *codifica*$_G$ bisogna rispettare tale aspettativa nel caso della *codifica*$_G$ dell'*MVP*$_G$:

- Le modifiche nei relativi *branch*$_G$ *feature*$_G$ riguardano solo le funzionalità segnalate e non verranno applicate all'*architettura*$_G$ e alla progettazione del documento di Specifica Tecnica. Nel caso in cui fosse necessario si concorda con il progettista come procedere e in caso affermativo di crea un ulteriore *branch*$_G$ a partire da develop che non riguarda la *feature*$_G$ in fase di implementazione.

Creazione di un *ticket*$_G$ per la codifica Per creare un *ticket*$_G$ legato allo sviluppo di funzionalità, si adotta la seguente procedura:

- identificare il *requisito*$_G$ che la funzionalità soddisfa (ad esempio: ROF34);
- identificare la parte architetturale che deve essere sviluppata per implementare la funzionalità descritta (ad esempio; deve essere inserita una nuova relazione all'interno del DB);
- identificare il risultato atteso dallo sviluppo;
- identificare l'assegnatario.

Il nome del *ticket*$_G$ sarà una combinazione dei primi due passi (ad esempio: ROF34/DB). All'interno della descrizione del *ticket*$_G$ si deve inserire il risultato atteso dallo sviluppo più eventuali accorgimenti o risorse per aiutare il programmatore.

Verifica del codice Il verificatore incaricato di verificare una funzionalità (corrispondente ad un *branch*$_G$ *feature*$_G$):

- Leggerà i cambiamenti segnalati dal programmatore nella *pull request*$_G$ e verificherà che:
    - La *feature*$_G$ richiesta sia stata implementata con successo;
    - La progettazione sia coerente con il documento di *specifica tecnica*$_G$ nel caso dell'*MVP*$_G$: ovvero che le classi implementate non si discostino dal documenti o che il diagramma ER relativo al database non sia stato modificato.

- In caso vengano soddisfatti tali requisiti, il verificatore approverà i cambiamenti; in caso contrario redigerà un *feedback*$_G$ nella *pull request*$_G$ con i cambiamenti da effettuare riportando la causa di eventuali errori e una possibile soluzione. Nel caso in cui la progettazione si discosti da quanto concordato, dovranno essere informati i progettisti in carica e discutere di tali cambiamenti con il programmatore prima di essere approvati.

Il responsabile è incaricato di accettere la *pull request*$_G$.
Nel momento in cui il responsabile ritiene che il codice possa essere rilasciato per la revisione allora viene creata una *pull request*$_G$ dal *branch*$_G$ develop verso il *branch*$_G$ main.

Creazione di un *ticket*$_G$ di verifica Per creare un *ticket*$_G$ legato alla verifica di una particolare funzionalità implementata, si adotta la seguente procedura:

1. identificare l'attività da verificare (la quale deve essere nello stato *in fase di verifica*);
2. inserire nella descrizione del *ticket*$_G$ il link all'attività che sta venendo verificata;
3. identificare l'assegnatario.

Il nome del *ticket*$_G$ dovrà rispettare la seguente *codifica*$_G$.

3.2.5 Strumenti

- ***Git***: *software*$_G$ utilizzato per il controllo di versione della documentazione e del codice prodotti;
- ***Git*$_G$*Hub***: piattaforma web utilizzata per il controllo di versione tramite *Git*$_G$.
- ***Jira***: piattaforma utilizzata per il ticketing.

3.3 Gestione della qualità

3.3.1 Descrizione

La gestione della qualità è un insieme di processi che hanno lo scopo di garantire che il *software*$_G$, gli artefatti e i processi nel ciclo di vita del progetto aderiscano degli standard di qualità rispetto a requisiti specificati al fine di soddisfare le aspettative del proponente e degli utenti finali.

3.3.2 Obiettivi

La gestione della qualità si propone di raggiungere i seguenti obiettivi:

- Realizzare un prodotto di qualità, in linea con le richieste del proponente;
- Ridurre al minimo i *rischi*$_G$ che potrebbero influire sulla qualità del prodotto;
- Rispettare il budget preventivato del progetto.

Gli strumenti utilizzati, per la gestione della qualità dei processi e del prodotto e per valutare il lavoro svolto, sono delle metriche definite nel documento di *Piano di Qualifica*.

3.3.3 Codifica delle metriche

Ogni metrica è identificata dal seguente formato di codice:

M[Tipo][Id]-[Acronimo]

Dove:

- **M** sta per "Metrica"
- **Tipo** può essere PC (per un *processo*$_G$) o PD (per un prodotto)
- **Id** rappresenta un identificativo all'interno di una metrica di un certo tipo
- **Acronimo** indica l'acronimo del nome della metrica utilizzata

Per ciascuna metrica vengono fornite delle descrizioni; inoltre per ogni tipo di *processo*$_G$ viene fornita una tabella avente: il codice della metrica, il nome della metrica, valori accettabili e valori preferibili.

3.4 Verifica

3.4.1 Descrizione

La verifica del *software*$_G$ è un *processo*$_G$ che valuta il prodotto durante le varie fasi del progetto, dalla progettazione alla *manutenzione*$_G$. Essa mira a garantire che il *software*$_G$ sia conforme alle aspettative e ai requisiti specificati fondandosi su criteri come coerenza, completezza e correttezza dei risultati.

3.4.2 Obiettivi

Il *processo*$_G$ di verifica si propone di raggiungere i seguenti obiettivi:

- Assicurarsi che il prodotto mantenga una buona qualità nel corso del suo sviluppo;
- Individuare errori e anomalie prima di proseguire con lo sviluppo del progetto.

Nel documento *"Piano di Qualifica*$_G$*"* vengono definiti gli obiettivi da raggiungere e i criteri di accettazione che saranno impiegati per condurre il *processo*$_G$ di verifica in modo accurato ed efficiente.

3.4.3 Analisi statica

L'analisi statica è una metodologia di verifica che prescinde dall'esecuzione del prodotto e che si basa su una revisione del codice e della documentazione. Lo scopo principale di questa analisi è quello di verificare l'assenza di difetti e la conformità ai requisiti e alle specifiche richieste.
L'analisi statica adotta comunemente due metodi di lettura:

- **Walkthrough**: si tratta di una tecnica collaborativa che coinvolge il verificatore e l'autore del prodotto e che consiste nel revisionare nel suo complesso il codice e la documentazione forniti, con una successiva discussione degli eventuali problemi trovati;

- **Inspection**: si tratta di una tecnica che consiste nel revisionare parti specifiche del codice e della documentazione attraverso liste di controllo (*checklist*) nel momento in cui si ha già un'idea di dove possano esserci possibili problemi in modo da intervenire tempestivamente e sistematicamente.

Nel documento *"Piano di Qualifica$_G$"* vengono definite delle liste di controllo in modo da applicare la tecnica dell'*Inspection*, preferibile a quella del *Walkthrough*.

3.4.4 Analisi dinamica

L'analisi dinamica è una metodologia di verifica che si basa sull'esecuzione del codice. Le tecniche principali utilizzate in questa fase sono i *test$_G$* (definiti nel documento di "*Piano di Qualifica*") finalizzati per individuare e verificare il comportamento del prodotto *software$_G$*.

3.4.5 Testing

L'obiettivo del testing è garantire il corretto funzionamento delle componenti, verificando che producano i risultati attesi. Questi *test$_G$* sono perciò responsabili di individuare eventuali anomalie di funzionamento.

3.4.5.1 Test di unità

I *test$_G$* di unità costituiscono un pilastro fondamentale nella verifica della correttezza delle singole unità di codice all'interno di un *sistema$_G$ software$_G$*. Essi sono progettati per isolare e verificare singoli componenti, come funzioni o metodi, garantendo che ciascuno di essi funzioni correttamente e produca risultati attesi in conformità alle specifiche.
L'obiettivo primario dei *test$_G$* di unità è garantire l'integrità e la correttezza di ogni singola unità di codice prima della sua integrazione con il resto del *sistema$_G$*.
Questo approccio consente di individuare e correggere eventuali errori o bug nelle fasi iniziali dello sviluppo, riducendo così il rischio di problemi più gravi nel *sistema$_G$* finale.
Durante l'implementazione dei *test$_G$* di unità, è possibile utilizzare oggetti simulati o parziali al fine di isolare l'unità di codice in esame dalle sue dipendenze esterne. Questo permette di verificare il comportamento dell'unità in con*test$_G$*i controllati e di garantire un'efficace separazione durante le fasi di *test$_G$*.
Inoltre, i *test$_G$* di unità sono progettati per raggiungere una copertura completa dei percorsi all'interno dell'unità di codice. Ciò significa che vengono creati appositamente *test$_G$* per attivare e verificare specifici percorsi nel codice, al fine di garantire una copertura completa e accurata delle varie parti dell'unità.

3.4.5.2 Test di sistema

L'obiettivo principale dei *test$_G$* di *sistema$_G$* è garantire che il *sistema$_G$* soddisfi i requisiti funzionali e non funzionali specificati durante la fase di progettazione e sviluppo. Essi mirano a convalidare le funzionalità del *sistema$_G$* nel con*test$_G$*o delle attività e dei processi previsti, verificando che il *sistema$_G$* si comporti correttamente e risponda in modo appropriato a tutte le richieste e le interazioni degli utenti.
Durante l'esecuzione dei *test$_G$* di *sistema$_G$*, vengono simulati scenari realistici e casi d'uso tipici che possono verificarsi durante l'utilizzo quotidiano del *sistema$_G$*.

Questo aiuta a identificare e risolvere eventuali difetti, errori o problemi di interoperabilità tra le diverse componenti del *sistema*$_G$.

3.4.5.3 Test di accettazione

L'obiettivo principale dei *test*$_G$ di accettazione è convalidare che il *software*$_G$ sia in grado di risolvere efficacemente i problemi e soddisfare le esigenze degli utenti.
Essi valutano il *sistema*$_G$ rispetto ai casi d'uso reali e alle specifiche funzionali concordate, verificando se il *software*$_G$ si comporta come previsto e se fornisce i risultati desiderati.

3.4.5.4 Formato dei test

La classificazione dei *test*$_G$ dei *test*$_G$ avviene seguendo il seguente formato:

T[Tipo]-[Codice]

dove il **Tipo** può essere "**U**" per i *test*$_G$ di unità,"**S**" per i *test*$_G$ di *sistema*$_G$, oppure "**A**" per i *test*$_G$ di accettazione.

3.4.5.5 Continuous Integration

L'integrazione continua è una pratica dello sviluppo di *software*$_G$ che consiste nell'automazione dell'integrazione del codice sorgente. Nel nostro progetto, questo concetto è stato messo in atto nella *repository*$_G$ dell'*MVP*$_G$, quando veniva creata una *pull request*$_G$ per integrare modifiche o migliorie al codice. Infatti quando viene creata una *pull request*$_G$, una *github*$_G$ action esegue i codici di *test*$_G$ dell'intero progetto.
una volta ottenuti gli esiti dei *test*$_G$ per ogni componente, i dati vengono inviati a *Coveralls*, che graficamente aggiorna il suo pannello di controllo per tenere traccia di tutte le percentuali di codice coperto, ed eventualmente capire dove effettuare *test*$_G$ più profondi per aumentare le percentuali di copertura.

3.4.6 Strumenti

- ***Coveralls***: è uno *strumento*$_G$ di analisi del codice e dei *test*$_G$ che fornisce informazioni sulla copertura del codice. Monitora l'esecuzione dei *test*$_G$ automatizzati e calcola la percentuale di righe di codice che sono eseguite dai *test*$_G$.

3.5 Validazione

3.5.1 Scopo ed Aspettative

La validazione è il *processo*$_G$ atto a verificare se un dato prodotto *software*$_G$ sia conforme ai requisiti e alle aspettative del cliente. Rappresenta una fase cruciale nello sviluppo *software*$_G$, e si concentra principalmente sui seguenti aspetti:

- **Conformità ai requisiti**: il prodotto deve soddisfare integralmente tutti i requisiti specificati dal cliente, così come descritti dal documento *Analisi dei Requisiti*;

- **Funzionamento corretto**: il prodotto deve funzionare correttamente, in conformità con la logica di progettazione;

- **Usabilità**: il prodotto deve essere di facile utilizzo per il target di utenza a cui si rivolge, e intuitivo;
- **Efficacia**: il prodotto deve soddisfare i bisogni del cliente.

3.5.2 Descrizione

Durante la fase di validazione, l'attenzione verrà concentrata principalmente sull'utilizzo dei $test_G$ precedentemente eseguiti durante l'attività di verifica, dettagliatamente normati nel documento *Norme di Progetto*. Con l'esecuzione del $test_G$ di accettazione, la validazione del prodotto potrà ritenersi conclusa.

4 Processi organizzativi

Ruoli di progetto In questa sezione viene riportata una breve descrizione dei ruoli e delle responsabilità dei membri di un gruppo dedicato allo sviluppo di un qualsiasi tipo di *project*.

4.1 Tutti i ruoli

Ogni ruolo dovrà tracciare le ore svolte per completare i $ticket_G$ definiti dal responsabile inserendo quanto tempo si è speso per completarla sul progetto di $jira_G$. Per la redazione di tutti i documenti eccetto il glossario tecnico si procede nel seguente modo:

- Si modifica il documento su overleaf
- Segnalare eventuali parole da aggiungere al glossario
- Aggiornare il $changelog_G$ riportando, in breve, la modifica apportata al documento
- Una volta scritto, spostare il $ticket_G$ in "da verificare" su jira

4.2 Responsabile di Progetto

Il Responsabile di Progetto è la figura professionale, punto di riferimento sia per il committente sia per il fornitore, con lo scopo di mediare tra le due parti. Assume la responsabilità delle decisioni del gruppo dopo averle approvate.
Le sue responsabilità includono:

- Approvare l'emissione della documentazione;
- Approvare l'offerta economica sottoposta al committente;
- Pianificare e coordinare le attività di progetto;
- Gestire le risorse umane;
- Studiare e gestire i $rischi_G$.
- Chiedere l'approvazione dei verbali alle persone esterne che hanno partecipato
- Assegnare le attività, tramite la funzione di tracking issues fornito da $Jira_G$, ai membri che le dovranno svolgere
- Gestire le *milestone* e fissarne di nuove, o modificare quelle attuali in base all'andamento del team

4.2.1 Gestione dello sprint

Inoltre, il responsabile dovrà gestire le tasks tramite $jira_G$, ovvero all'inizio di ogni $sprint_G$ dovrà:

1. Individuare delle attività da svolgere nel periodo di sprint
2. Determinare la durata dello sprint

3. Creare dei tickets, associati alle attività, su *jira*$_G$ impostando un tempo previsto per svolgere l'attività (le attività di verifica vengono aggiunte anche durante il periodo di *sprint*$_G$)
4. Decidere i ruoli dei membri ed associarli alle attività che ognuno dovrà svolgere
5. Stilare il *preventivo*$_G$ sul file excel apposito facendo la somma delle ore stimate per concludere le attività
6. Creare un nuovo periodo nel PDP nella sezione "Pianificazione" e nella sottosezione planning inserire le attività da svolgere e il *preventivo*$_G$ (creato nel punto precedente tramite excel)

Durante lo *sprint*$_G$ il responsabile di progetto dovrà:

1. Inserire nel PDP (nella sezione "Review") le attività completate
2. Aggiornare il *consuntivo*$_G$ nel file excel con le attività completate
3. Inserire gli eventuali *rischi*$_G$ nella sezione "Analisi dei *rischi*$_G$"

Infine, alla fine del periodo:

1. Inserire il *consuntivo*$_G$ nella sezione Review del Piano di Progetto
2. Completare la sezione "Retrospective", analizzando le criticità e i *rischi*$_G$ e su come si potrebbero risolvere

4.2.2 Verifica dei documenti

Il responsabile, quando un documento viene verificato, lo inserirà nella *repository*$_G$ Project14 nel *branch*$_G$ develop, aggiornando il file README. Quindi se il file che viene verificato è un documento (tranne il glossario) si deve:

1. Scaricare il sorgente del documento tramite *overleaf*$_G$ (si deve andare in progetto > menù > sorgente)
2. Inserire la cartella in develop, eventualmente sostituendola a quella presente
3. Utilizzare il seguente comando per far partire l'automazione: python3 entry_point_automazioni.py, si otterrà un file *latex*$_G$ con i *riferimenti*$_G$ al glossario
4. Tramite LatexWorkshop, per l'installazione , (o strumenti simili) generare il file *pdf*$_G$ corrispondente
5. Inserire il file *pdf*$_G$ nella cartella documenti aggiornando il README
6. Segnare sia il *ticket*$_G$ della stesura del documento e sia della verifica del documento in "completato" tramite jira

Se invence il documento approvato è il glossario allora si procede nel seguente modo:

1. Si approva la *pull request*$_G$ associata in develop
2. Si inseriscono tutti i sorgenti dei documenti presenti su *overleaf*$_G$ (se approvati) tranne i verbali

3. Si utilizza l'automazione: python3 entry_point_automazioni.py cosicché vengono generati i file *latex*$_G$ dei documenti e del glossario

4. Utilizzando LatexWorkshop per generare i *pdf*$_G$ e si inseriscono nella *repository*$_G$ su develop

5. Infine si aggiorna il README

4.2.3 Verifica del codice

Invece, se invece l'attività riguardava al codice allora bisogna approvare o rifiutare la pull request

4.3 Amministratore di Progetto

L'Amministratore di Progetto è responsabile delle procedure di controllo e amministrazione dell'ambiente di lavoro, con piena responsabilità sulla capacità operativa e sull'efficienza. Questo ruolo è cruciale per garantire che il progetto di sviluppo *software*$_G$ proceda senza intoppi, rispettando i tempi e le risorse assegnate.
In particolare, si occupa di:

- **Ricercare, studiare e mettere in opera risorse per migliorare l'ambiente di lavoro, automatizzandolo quando possibile**: L'Amministratore di Progetto identifica nuove *tecnologie*$_G$, strumenti e pratiche che possono ottimizzare il flusso di lavoro del team di sviluppo. L'obiettivo è creare un ambiente di lavoro efficiente e innovativo, che faciliti la collaborazione e aumenti la produttività.

- **Risolvere problemi legati alla gestione dei processi**: Quando emergono problemi nei processi di sviluppo, l'Amministratore di Progetto interviene per identificare le cause e implementare delle soluzioni.

- **Salvaguardare la documentazione di progetto**: L'Amministratore di Progetto garantisce che tutta la documentazione sia accurata, aggiornata e facilmente accessibile ai membri del team. Questo aiuta a mantenere la coerenza e a facilitare la comunicazione tra tutte le parti coinvolte nel progetto.

- **Effettuare il controllo di versioni e configurazioni del prodotto *software*$_G$**: Un'efficace gestione delle versioni e delle configurazioni è fondamentale per prevenire problemi d'incompatibilità e garantire che tutte le modifiche al *software*$_G$ siano tracciate e documentate. L'Amministratore di Progetto sovrintende l'uso di sistemi di controllo di versione.

- **Redigere e attuare i piani e le procedure per la gestione della qualità**: La qualità del prodotto *software*$_G$ è una priorità. L'Amministratore di Progetto sviluppa piani di qualità che definiscono gli standard e le metriche di qualità da raggiungere.

4.4 Analista

L'Analista è una figura con maggiori competenze riguardo al dominio applicativo del problema. Questo ruolo è cruciale per garantire che il team di sviluppo comprenda appieno le esigenze degli utenti e i requisiti del progetto, traducendoli in specifiche tecniche chiare e dettagliate.
Le sue responsabilità includono:

- **Studiare il problema e il relativo contesto applicativo**: L'Analista investiga a fondo il problema da risolvere e il contesto in cui il *software*$_G$ verrà utilizzato. Questo include l'identificazione delle principali sfide e l'interazione con gli *stakeholder* per raccogliere informazioni critiche.

- **Comprendere il problema e definire la complessità e i requisiti**: Una volta raccolte le informazioni iniziali, l'Analista deve sintetizzare e comprendere il problema in modo completo. Questo comporta la definizione dei requisiti funzionali, la valutazione della complessità del progetto e la prioritizzazione delle esigenze degli utenti.

- **Redigere il documento *Analisi Dei Requisiti***: L'Analista documenta i requisiti del progetto in un documento formale, l'Analisi dei Requisiti. Questo documento funge da guida per tutto il team di sviluppo e garantisce che tutti i membri abbiano una comprensione condivisa degli obiettivi del progetto.

- **Studiare i casi d'uso e redigere il loro relativo schema *UML*$_G$**: L'Analista crea e analizza i casi d'uso, che descrivono come gli utenti interagiranno con il *sistema*$_G$. Questi casi d'uso sono rappresentati graficamente mediante schemi *UML*$_G$, che aiutano a visualizzare le interazioni tra utenti e *sistema*$_G$ e a identificare eventuali lacune o problemi nelle specifiche.

- **Aggiornare, in base alle attività definite dai tickets, i documenti *Norme di Progetto* e *Piano di Qualifica***: L'Analista è responsabile di mantenere aggiornati anche questi documenti di progetto. Questo assicura che la documentazione rifletta sempre lo stato attuale del progetto e che le modifiche siano tracciate e gestite in modo appropriato.

4.5 Progettista

Il Progettista gestisce gli aspetti tecnologici e tecnici del progetto, assicurandosi che le soluzioni proposte siano efficienti, sostenibili e in linea con i requisiti definiti dall'Analista. Questo ruolo è fondamentale per tradurre le esigenze del progetto in un *design*$_G$ tecnico concreto e funzionale.
In particolare, si occupa di:

- **Effettuare scelte riguardanti gli aspetti tecnici e tecnologici del progetto**: Il Progettista seleziona le *tecnologie*$_G$, gli strumenti e le metodologie più appropriati per lo sviluppo del progetto. Questa scelta è influenzata dalla necessità di bilanciare l'efficacia, l'efficienza e i costi del progetto, garantendo nel contempo che le soluzioni tecniche adottate siano all'avanguardia e adatte al contesto applicativo.

- **Definire un'*architettura*$_G$ del prodotto da sviluppare che miri all'economicità e alla manutenibilità a partire dal lavoro svolto dall'Analista**: Il Progettista crea un'*architettura*$_G$ del *sistema*$_G$ che soddisfi i requisiti definiti dall'Analista, ponendo particolare attenzione alla manutenibilità, scalabilità e sostenibilità del progetto. Questo include la progettazione di componenti *software*$_G$ modulari e riutilizzabili, che facilitano l'aggiornamento e l'espansione del *sistema*$_G$ nel tempo.

- **Redigere la *Specifica Tecnica* e la parte pragmatica del *Piano di Qualifica***: Il Progettista documenta dettagliatamente le scelte tecniche e le architetture proposte nel documento *Specifica Tecnica*. Inoltre, contribuisce alla stesura della parte pragmatica del documento*Piano di Qualifica*, che descrive come verranno verificate e validate le soluzioni tecniche implementate, assicurandosi che soddisfino gli standard di qualità previsti dal progetto.

4.6 Verificatore

Il Verificatore è responsabile della sorveglianza sul lavoro svolto dagli altri componenti del gruppo, sulla base delle proprie competenze tecniche, esperienza e conoscenza delle norme.
In particolare, si occupa di:

- Esaminare i prodotti in fase di revisione, con l'ausilio delle tecniche e degli strumenti definiti nel presente documento;
- Verificare la conformità dei prodotti ai requisiti funzionali e di qualità;
- Verificare i documenti segnalando eventuali errori.
- Caricare i verbali, dopo averli verificati, all'interno della *Repository*$_G$ *Github*$_G$ nel ramo *develop*.

Si precisa che la verifica dei documenti deve essere svolta tramite controllo della sintassi e semantica del documento in questione e, inoltre, deve assicurarsi che quanto definito dal documento sia corretto e congruente con altri eventuali documenti. Infine il verificatore può segnalare elementi da aggiungere nel *Piano di Progetto*$_G$ che possono tornare utili agli altri verificatori.

4.6.1 Verifica dei documenti

Per completare la verifica di un documento, tranne il glossario, si procede nel seguente modo:

- Verificare il documento su overleaf
- Controllare eventuali *riferimenti*$_G$ al glossario (si ricorda che le parole del glossario sono case sensitive)
- Se il documento viene approvato allora bisogna segnare come "verificato" sia il *ticket*$_G$ relativo alla verifica del documento sia il *ticket*$_G$ della stesura del documento presenti su jira

Per la verifica del glossario si faccia riferimento alla sezione 3.1.12 .

4.6.2 Verifica del codice

Il verificatore *NON* deve spostare i *ticket*$_G$ in completato che riguardano ai programmatori in quanto è compito del responsabile una volta approvata la *pull request*$_G$.

4.7 Programmatore

Il Programmatore è incaricato della *codifica*$_G$ del progetto e delle componenti di supporto che verranno utilizzate per eseguire prove di verifica e di validazione del prodotto.
In particolare, si occupa di:

- Implementare la "*specifica tecnica*$_G$" redatta dal Progettista;
- Scrivere un codice pulito e facilmente mantenibile che rispetti le norme definite nel presente documento;
- Realizzare gli strumenti per la verifica e la validazione del *software*$_G$;
- Redigere il "*Manuale Utente*$_G$" relativo alla propria *codifica*$_G$;
- Redigere i verbali delle riunioni interne ed esterne del team.

4.7.1 Gestione del repository

Inoltre, di seguito viene specificato come gestire la *repository*$_G$ per l'implementazione di nuove *feature*$_G$:

1. Tramite *jira*$_G$ creare un nuovo *branch*$_G$ associato al *ticket*$_G$, come viene descritto nella sezione gestione del repository
2. Eseguire il *push*$_G$ una volta completata l'attività
3. Creare una *pull request*$_G$ dal nuovo *branch*$_G$ verso il *branch*$_G$ develop (tramite jira/github)
4. Una volta eseguito si sposta il *ticket*$_G$ nella sezione "da verificare"

4.8 Gestione di progetto

4.8.1 Allineamento organizzativo

Per l'allineamento e il coordinamento delle attività sono stati scelti due tipi di comunicazione:

- Interne: coinvolge tutti i membri del gruppo;
- Esterne: coinvolge proponenti e committenti.

4.8.2 Comunicazioni interne

Le comunicazioni interne avvengono tramite le applicazioni:

- *Telegram*: permette una comunicazione veloce tra i membri del gruppo e viene utilizzato principalmente per organizzare incontri interni o discutere di eventuali quesiti.
- *Discord*: viene utilizzato dai membri del gruppo per tenere incontri interni, in quanto dispone di un canale vocale; inoltre, permette la condivisione di contenuti multimediali in streaming video.

4.8.3 Comunicazioni esterne

Le comunicazioni esterne vengono gestite dal Responsabile di Progetto tramite l'utilizzo dei seguenti canali:

- *Posta elettronica*$_G$: tramite l'indirizzo e-mail del gruppo *ramtastic6@gmail.com*.
- *Telegram*: permette di instaurare un canale veloce di comunicazione con la proponente.

4.9 Gestione organizzazione del lavoro

4.9.1 Modello di sviluppo

Il gruppo, al fine di minimizzare ritardi e massimizzare lo svolgimento delle proprie attività, ha deciso di implementare il *framework*$_G$ *Scrum*, che permette di dividere il tempo di lavoro in intervalli piccoli, frammentati all'interno di un periodo di circa due settimane, definito come "*sprint*". In particolare, all'interno di questo, vengono svolte un numero di attività commisurate che devono essere completate entro i suoi termini. Individuiamo dunque una serie di fasi:

- **sprint planning**: in questa fase, si discutono gli obbiettivi da raggiungere all'interno dello *sprint*; vengono quindi pianificate le attivitá da svolgere durante lo *sprint* in funzione di questi, facendo attenzione alle disponibilitá di ogni membro e preparando un *preventivo*$_G$ per il periodo; è in questa fase, inoltre, che viene svolto il cambio dei ruoli. Lo *sprint planning* consiste in una riunione tenuta tramite *Discord* che si svolge all'inizio dello *sprint* e richiede la partecipazione di tutti i membri del gruppo.

- **sprint review**: in questa fase, si discutono gli obiettivi da raggiunti nello *sprint*. Alla fine, dovrebbe essere prodotto almeno un incremento, cioè un *software*$_G$ utilizzabile. In particolare, in questa fase vengono definiti a *consuntivo*$_G$ le risorse impiegate commisurate agli obiettivi, questi ultimi suddivisi in raggiunti e non raggiunti, capendo cosa va migliorato e cosa non è stato fatto per poter aggiungere ulteriori obiettivi per lo *sprint* successivo;

- **sprint retrospective**: fase che conclude definitivamente lo *sprint* appena svolto, avente l'obiettivo di valutarne l'andamento generale e cercando di capire cosa è stato fatto bene e cosa può essere migliorato. In questo modo, è meglio definire come ripianificare le attività, decidendo come iniziare/continuare/concludere attività presenti o future da realizzare.

4.9.2 Rotazione dei ruoli

É prevista una rotazione dei ruoli all'interno del gruppo a cadenza periodica. L'attribuzione dei ruoli viene svolta secondo i seguenti criteri:

- Equità;
- Disponibilità;
- Assenza di conflitti.

4.9.3 Gestione delle attività

Per gestire le attività da istanziare all'interno del *framework*$_G$ *Scrum*, è stato deciso di utilizzare *Jira*. Un'attività, definita come *ticket* all'interno di *Jira*, ha un ciclo di vita; esso si compone di quattro stati:

- **Da completare**: l'attività è stata istanziata e assegnata ad un membro del gruppo; è stata inoltre definita una stima in termini temporali associata ad essa.

- **In corso**: l'attività è entrata nella fase di svolgimento; all'interno di questa fase, il membro al quale è stata assegnata ha il compito di dover inserire le informazioni di tracciamento temporale associate al *ticket*; per fare ciò, basta selezionare un *ticket* e andare a modificare il campo *Tracciamento temporale*, inserendo il tempo impiegato fino ad ora per svolgere l'attività.

- **In fase di verifica**: durante questa fase, il Verificatore si assicura che l'attività sia stata svolta correttamente e secondo dei criteri di qualità attesa (al momento, ancora da specificare).

- **Completata**: l'attività è stata verificata e approvata dal Responsabile di Progetto.

Il passaggio di un *ticket* da uno stato all'altro è responsabilità del membro al quale è stato assegnato, eccetto per lo stato di completamento; infatti, spetta al Responsabile di Progetto decidere quando un'attività risulti o meno completata.

4.10 Infrastruttura

Strumenti per la gestione delle attività di progetto e procedure legate ad essi.

4.11 Miglioramento

Durante lo svolgimento delle attività e successiva stesura dei documenti, il gruppo si impegna ad operare secondo il *principio di miglioramento continuo*, al fine di individuare facilmente attività, ruoli e possibili miglioramenti, cercando nuove o diverse soluzioni alle problematiche insorte.

4.12 Formazione

Al fine di operare continuativamente un miglioramento sulle attività svolte e proseguire nel mantenimento corretto delle attività in modo asincrono, occorre da parte di tutti i membri del gruppo lo studio in autonomia delle *tecnologie*$_G$ e delle modalità operative presenti, al fine di velocizzare l'apprendimento degli strumenti utilizzati. Si listano, al fine di completezza, alcune documentazioni utilizzate durante lo sviluppo, sia a livello documentale che organizzativo:

- *Github*: https://docs.github.com/
- *Jira*: https://confluence.atlassian.com/jira
- *Git*: https://docs.github.com/en/get-started/using-git/about-git
- *Scrum*: https://scrumguides.org/scrum-guide.html