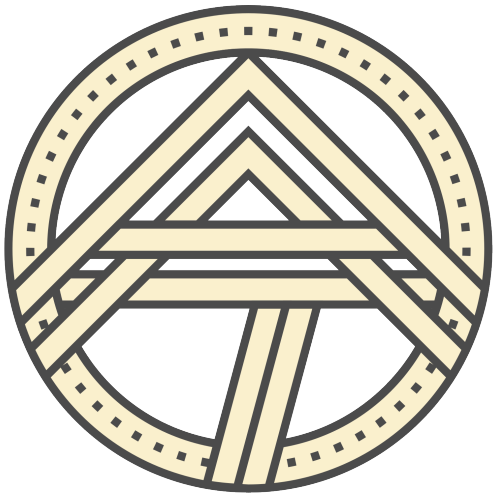
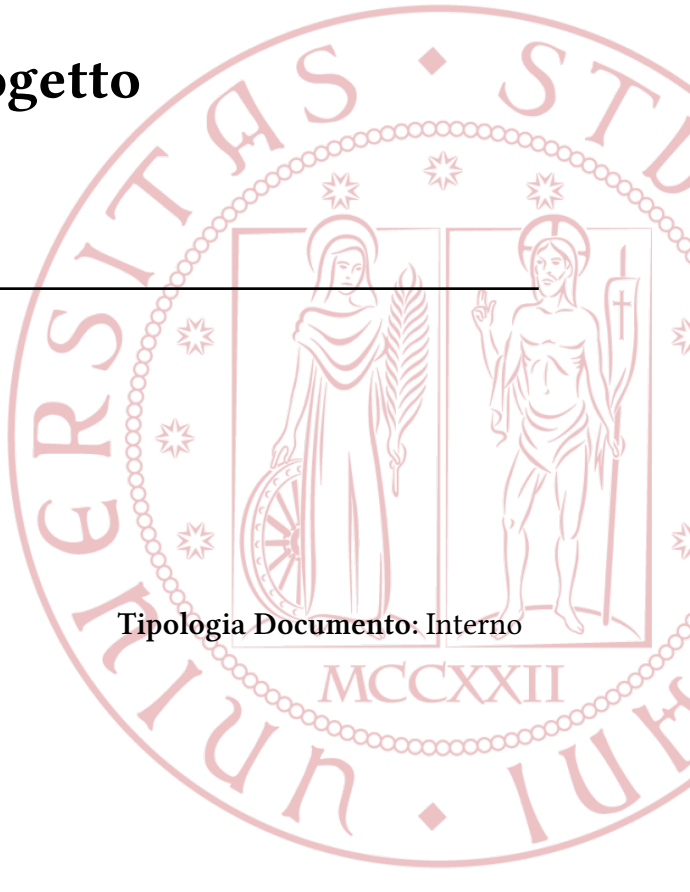


ARCHI7ECHS

Archi7echs - archi7echs@gmail.com

Progetto di **Ingegneria del Software**
A.A. 2024/2025

Norme di Progetto



Autore: Il team

Ultima Modifica: 22/04/2025

Tipologia Documento: Interno

Tabella delle revisioni

Rev.	Data	Descrizione	Elaborazione	Verifica
2.0.0	22-04-2025	Revisione per incontro PB	Pietro Valdagno	Gabriele Checchinato, Giovanni Salvò
1.2.0	21-03-2025	Ampliamento sezione processi di supporto	Gabriele Checchinato, Pietro Valdagno	Giovanni Salvò, Giacomo Pesenato
1.1.1	21-03-2025	Modifica sezione riferimenti, documentazione da consegnare e sviluppo	Gabriele Checchinato	Giovanni Salvò, Pietro Valdagno
1.1.0	12-03-2025	Modifica sezioni da stile narrativo a procedurale. Aggiunta sezione relativa al versionamento. Aggiunti riferimenti alla fase PB. Aggiunto riferimento a fase di codifica - utilizzo di Git e della repo	Francesco Pozzobon	Giacomo Pesenato, Gioele Scandaletti
1.0.0	09-02-2025	Revisione per incontro RTB	Francesco Pozzobon	Pietro Valdagno, Leonardo Lucato
0.18.1	31-01-2025	Aggiunta termini glossario	Gioele Scandaletti	Francesco Pozzobon, Leonardo Lucato
0.18.0	29-01-2025	Aggiunta sezione sviluppo e sottosezioni validazione, configurazioni, qualità, miglioramento e formazione	Giovanni Salvò	Leonardo Lucato, Giacomo Pesenato
0.17.1	18-01-2025	Aggiunta metrica Cost Variance	Pietro Valdagno	Leonardo Lucato, Giacomo Pesenato
0.17.0	14-01-2025	Riscrittura relazione di estensione con extension points	Gioele Scandaletti	Leonardo Lucato, Francesco Pozzobon
0.16.0	12-01-2025	Aggiunta sottosezione Diagramma di Gantt	Gioele Scandaletti	Giacomo Pesenato, Francesco Pozzobon
0.15.0	12-01-2025	Aggiunta sottosezione Management - Gestione immagini	Gioele Scandaletti	Giacomo Pesenato, Francesco Pozzobon
0.14.0	10-01-2025	Stesura metriche di qualità e riscrittura introduzione e scopo della sezione fornitura	Pietro Valdagno	Giacomo Pesenato, Francesco Pozzobon
0.13.2	09-01-2025	Riorganizzazione di alcune sezioni	Leonardo Lucato	Giacomo Pesenato, Francesco Pozzobon
0.13.1	06-01-2025	Fix - correzione ortografico Introduzione	Francesco Pozzobon	Giacomo Pesenato, Pietro Valdagno
0.13.0	04-01-2025	Stesura standard di qualità	Pietro Valdagno	Leonardo Lucato, Francesco Pozzobon
0.12.0	02-01-2024	Diagrammi UML (casi d'uso e classi). Documentazione da consegnare	Leonardo Lucato	Giacomo Pesenato, Francesco Pozzobon
0.11.0	20-12-2024	Stesura acronimi e abbreviazioni. Ristrutturazione versionamento	Francesco Pozzobon	Giovanni Salvò, Pietro Valdagno
0.10.0	19-12-2024	Stesura gestione e analisi ore lavorative	Gabriele Checchinato	Gioele Scandaletti, Pietro Valdagno
0.9.0	19-12-2024	Stesura processi organizzativi-gestione dei processi e correzioni	Francesco Pozzobon	Gioele Scandaletti, Pietro Valdagno

0.8.0	17-12-2024	Stesura comunicazione interna del team	Francesco Pozzobon	Giovanni Salvò, Pietro Valdagno
0.7.0	16-12-2024	Stesura norme tipografiche	Gabriele Checchinato	Giovanni Salvò, Pietro Valdagno
0.6.0	15-12-2024	Stesura processi primari-comunicazioni con proponente e strumenti	Francesco Pozzobon	Giovanni Salvò, Pietro Valdagno
0.5.0	15-12-2024	Redatta sezione Gestione dell'assegnazione ruoli	Giovanni Salvò, Francesco Pozzobon	Gioele Scandaletti, Pietro Valdagno
0.4.1	10-12-2024	Fix sezione Verifica e Revisione della documentazione	Giovanni Salvò	Pietro Valdagno, Gabriele Checchinato
0.4.0	26-11-2024	Redatta gestione della board e istruzioni per la redazione/verifica dei documenti	Leonardo Lucato	Gabriele Checchinato
0.3.0	25-11-2024	Redatta sottosezione Documentazione	Francesco Pozzobon	Giovanni Salvò
0.2.0	24-11-2024	Redatta sezione Introduzione	Leonardo Lucato	Gabriele Checchinato
0.1.0	24-11-2024	Redatta la suddivisione del documento	Francesco Pozzobon	Gabriele Checchinato

Indice

1) Introduzione

1.1) Finalità del documento

L'obiettivo del documento è quello di definire le linee guida del gruppo per garantire un lavoro, fortemente asincrono, uniforme, coerente e di qualità. Per gestire il prodotto, che comprende software e documentazione, è necessario adottare un approccio strutturato al ciclo di vita$_G$.
Tale documento è redatto secondo lo standard ISO 12207:1995$_G$, il quale identifica i processi di un ciclo di vita$_G$ di un software, secondo una struttura modulare con relativa responsabilità su ciascun processo.

1.2) Glossario

All' interno$_G$ del documento saranno spesso utilizzati degli acronimi o termini tecnici per semplificare la scrittura e la lettura. Per garantire che quanto scritto sia comprensibile a chiunque, è possibile usufruire del *glossario*. Tutte le parole consultabili nel glossario saranno identificate da una «G» in colore blu. Facendo click sul collegamento si aprirà una scheda del browser con il glossario

1.3) Riferimenti

Il documento è stato redatto con riferimento alla seguente documentazione.

1.3.1) Riferimenti informativi

- Riferimento al capitolato$_G$ 5 di ***Sanmarco Informatica SPA - 3Dataviz***: https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C5.pdf - *Ultimo accesso al documento 22/11/2024*
- Riferimento alle slide IS: ***Gestione di progetto***: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T04.pdf - *Ultimo accesso al documento 01/02/2025*
- Riferimento alle slide IS: ***Regolamento del progetto didattico***: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf - *Ultimo accesso al documento 12/12/2024*

1.3.2) Riferimenti normativi

- Riferimento alle slide IS: ***Processi di ciclo di vita***: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T02.pdf - Sezione sullo standard ISO 12207:1995$_G$ - *Ultimo accesso al documento 22/01/2025*

2) Processi Primari

2.1) Fornitura

2.1.1) Introduzione

Secondo lo standard ISO/IEC 12207:1995$_G$, la fornitura viene definita come un insieme strutturato di attività$_G$ e processi per la gestione e lo sviluppo del progetto e quindi per realizzare il prodotto software richiesto dal committente.

2.1.2) Scopo

Il processo si concentra sul monitoraggio e sulla gestione delle attività$_G$ svolte dal team durante le varie fasi del progetto, dalla concezione iniziale fino alla consegna, assicurandosi che il prodotto finale rispetti i requisiti concordati con il committente, oltre a essere realizzato entro i tempi e i costi stabiliti. In questo modo viene garantita una visione completa e coerente della gestione delle attività$_G$ durante l'intero ciclo di vita$_G$ del progetto.

2.1.3) Comunicazione con l'azienda proponente

Le comunicazioni con Sanmarco Informatica, azienda proponente$_G$ del progetto, avvengono principalmente via Google Chat. Alex Beggiato, System Architect Team Leader, si rende disponibile a rispondere a eventuali domande o dubbi bloccanti durante il periodo$_G$ secondo la modalità di cui sopra oppure attraverso una riunione dedicata via Google Meet.
Gli incontri di Stato Avanzamento Lavori, SAL$_G$, vengono fissati di volta in volta a fine periodo$_G$, con l'impegno di non superare due settimane tra un incontro e l'altro, salvo esplicite motivazioni.
Durante tale incontro, con relativo verbale esterno$_G$, il responsabile$_G$ del periodo$_G$ in corso rendiconta, in via generale, quanto svolto lasciando poi la parola ai diretti interessati per esposizione dettagliata del lavoro svolto e chiarimento di dubbi.

2.1.4) Documentazione da consegnare

In questa sezione vengono indicati i documenti che saranno consegnati all'azienda proponente *Sanmarco Informatica* e ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.

2.1.4.1) Analisi dei Requisiti

All' interno$_G$ vengono definite le funzionalità$_G$ che la nostra webapp deve supportare, in modo da garantire un ottimo studio preliminare approfondito del progetto. Il documento deve contenere:

- **Casi d'uso**: che rappresentano in modo formale le funzionalità$_G$ di un sistema, illustrando le attività$_G$ svolte durante un'interazione
- **UML$_G$ casi d'uso**: che rappresentano in modo grafico/visivo l'interazione tra un attore e uno o più casi d'uso
- **Requisiti**: ovvero l'insieme delle funzionalità$_G$ richieste e quelle proposte in sede interna al gruppo. È dunque tutto quello che è stato pensato per far funzionare al meglio la webapp

2.1.4.2) Piano di Progetto

Documento fondamentale per il gruppo, che permette di eseguire delle buone retrospettive, con un automiglioramento sia per il breve che per il lungo termine. All' interno$_G$ del documento ci devono essere i seguenti punti:

- **Analisi dei rischi**: ovviamente è indispensabile per una buona pianificazione di ogni singolo periodo$_G$, per avere già delle tecniche e strategie da applicare quando

- **Informazioni del progetto**: ovvero tutte quelle informazioni che è bene tenere traccia (e anche modificare in futuro).
  - Tempi di consegna previsti
  - Costi previsti
  - Struttura della pianificazione di un periodo$_G$
  - Struttura di quanto accaduto nell'effettivo durante un periodo$_G$
- **Lista dei periodi**: ovvero l'insieme tra il preventivo e il consuntivo, di quello che è accaduto, quello che è andato quanto pianificato e soprattutto quello che invece non era stato pianificato, fondamentale per l'automiglioramento

2.1.4.3) Piano di qualifica

Documento che serve al team per descrivere come è stata garantita l'efficienza durante il progetto. Questo serve per garantire ai committenti, all'azienda proponente e al team, che ci sono dei processi selezionati solo allo scopo di verificare che quanto fatto sia di ottima qualità, che sono state investite quantità di risorse ottime e che il prodotto rispetti le aspettative richieste. All' interno$_G$ del documento dunque devono essere presenti:

- **Test del prodotto**: per garantire che il prodotto soddisfi quanto proposto dai committenti e dall'azienda proponente è necessario eseguire una serie di test che, solo dopo il loro test positivo, è possibile dichiarare un prodotto «soddisfacente»
  - **Test di sistema**
  - **Test di integrazione**
  - **Test di unità**$_G$
- **Metriche per garantire la qualità dei processi**: verifica e validazione$_G$ dei processi, dunque come garantire che ogni processo abbia il risultato atteso (buona qualità con un quantitativo di risorse investite ottimo)
- **Metriche per garantire la qualità del prodotto**: verifica e validazione$_G$ per garantire che il prodotto sia conforme a tutti gli obbiettivi di qualità

2.1.4.4) Specifica Tecnica

Il documento rappresenta una risorsa tecnica essenziale per la comprensione approfondita dell' architettura$_G$ e delle scelte progettuali adottate nel sistema in sviluppo. Il suo obiettivo principale è fornire una descrizione dettagliata delle componenti software e delle strategie di progettazione seguite dal team.

In particolare, vengono trattati due aspetti chiave:

- **Architettura$_G$ implementativa**: descrive un'analisi dettagliata delle strutture software, dei design pattern utilizzati e delle logiche di implementazione adottate per garantire scalabilità, manutenibilità ed efficienza del sistema.
- **Architettura$_G$ di deployment**: descrive la configurazione dell'infrastruttura, gli ambienti di esecuzione e le modalità di distribuzione del software.

Il documento include inoltre la lista delle tecnologie utilizzate e i diagrammi UML$_G$ (ad es. diagrammi delle classi e dei componenti) per supportare una comprensione chiara e strutturata del software. Esso quindi funge anche da guida per lo sviluppo, per la manutenzione e l'evoluzione del progetto, garantendo al contempo una copertura completa dei requisiti definiti nel documento Analisi dei Requisiti$_G$.

2.1.4.5) Manuale Utente

Il presente manuale è progettato per offrire un supporto completo agli utenti nell'utilizzo del software$_G$, guidandoli attraverso le sue funzionalità in modo chiaro e dettagliato. L'obiettivo principale è consentire agli utenti di comprendere e sfruttare appieno le potenzialità del sistema, garantendo un'esperienza d'uso efficiente e intuitiva.

Il manuale si concentra principalmente sulle istruzioni operative per l'utilizzo quotidiano. Vengono fornite spiegazioni dettagliate sui principali flussi operativi, sulle modalità di interazione con l'interfaccia e sulle best practice per un uso ottimale del sistema.

2.1.4.6) Lettera di presentazione

Quando si deve consegnare quanto fatto (RTB$_G$ o PB$_G$) ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin è necessario farlo tramite un'invio di una mail **priva di allegati**, ma con un solo puntatore alla *Lettera di presentazione* che deve essere in repo$_G$. È dunque un documento che attesta il «completamento» di una delle due parti del progetto, dichiarando di essere pronti alla revisione$_G$ di tutta la documentazione/materiale fatta fino a quel momento. All' interno$_G$ della *Lettera di presentazione* ci deve essere:

- **Introduzione**: per indicare lo scopo del documento
- **Link alla documentazione**: un puntatore che riporta alla repo$_G$ del gruppo, dov'è possibile reperire tutta la documentazione
- **Lista di documenti presenti**: ovvero l'insieme dei verbali sia interni$_G$ che esterni$_G$ che il gruppo ha redatto durante il periodo$_G$, la documentazione «interna» e la documentazione «esterna».
- **Lista dei componenti del gruppo**: una semplice tabella che indica il nome e cognome accompagnati dalla matricola di ogni singolo componente del gruppo

2.2) Strumenti

Sono attivi i seguenti strumenti e canali di comunicazione a disposizione dei membri del team:

- **Gruppo Telegram**$_G$ per le comunicazioni rapide ed informali
- **Canale Discord**$_G$ per le riunioni del gruppo in videoconferenza e le comunicazioni ufficiali, organizzate nei relativi sotto-canali
- **Gmail** per le comunicazioni ufficiali con il committente
- **Google Chat** per le comunicazioni con l'azienda proponente
- **Google Meet** per le riunioni in conferenza con l'azienda proponente
- **Google Drive e suite Google Documenti** per l'archiviazione e la modifica dei file condivisi del gruppo, quali:
  - Foglio appunti riunioni
  - Foglio ore condiviso

2.3) Sviluppo

2.3.1) Introduzione

Secondo lo standard ISO/IEC 12207:1995$_G$, lo sviluppo viene definito come un insieme strutturato di attività$_G$:

1. **Implementazione del processo**
2. **Analisi dei requisiti di sistema**
3. **Progettazione architetturale del sistema**
4. **Analisi dei requisiti software**
5. **Progettazione architetturale del software**
6. **Progettazione dettagliata del software**
7. **Codifica e testing del software**
8. **Integrazione$_G$ del software**
9. **Test di qualifica del software**
10. **Integrazione$_G$ di sistema**
11. **Test di qualifica del sistema**
12. **Installazione del software**
13. **Supporto all'accettazione del software**

Queste attività garantiscono che il ciclo di vita del software segua un processo strutturato e controllato, assicurando la qualità e la conformità ai requisiti stabiliti.

2.3.1.1) Implementazione del processo

Questa fase costituisce il punto di partenza del ciclo di vita$_G$ del software$_G$, stabilendo il modello di sviluppo secondo lo standard ISO/IEC 12207:1995$_G$. Si definiscono le responsabilità dei vari attori coinvolti, includendo sviluppatori, responsabili della documentazione$_G$ e del controllo qualità. Viene scelto un framework$_G$ metodologico che guidi ogni fase del processo e si identificano gli standard$_G$ e le best practice da applicare. Strumenti$_G$, linguaggi di programmazione e metodi vengono selezionati per supportare le attività$_G$ successive. Si redige un piano operativo che definisca tappe, risorse e scadenze. La documentazione iniziale garantisce la tracciabilità delle decisioni. Infine, revisioni periodiche permettono di affinare il modello e rispondere ai requisiti emergenti.

2.3.1.2) Analisi dei requisiti di sistema

Questa fase si concentra sull'identificazione dettagliata delle esigenze a livello di sistema$_G$ e sulla definizione dei requisiti$_G$. Si esaminano i contesti operativi, le condizioni ambientali e le prestazioni attese, considerando aspetti quali sicurezza, usabilità e affidabilità. Vengono analizzati i vincoli tecnici e organizzativi, e la raccolta dei requisiti include input da fonti interne ed esterne. Ogni requisito è documentato in modo chiaro e verificabile, assicurando la tracciabilità attraverso strumenti specifici. Revisioni periodiche garantiscono coerenza e fattibilità tecnica, fornendo la base per la progettazione architetturale del sistema.

2.3.1.3) Progettazione architetturale del sistema

In questa fase viene definita una visione d'insieme dell'intero sistema$_G$, individuando i principali componenti hardware$_G$ e software$_G$ e le relative operazioni manuali. Ogni requisito viene allocato in specifici elementi, mentre le interfacce tra i componenti vengono chiaramente delineate per garantire scalabilità e modularità. La documentazione prodotta descrive la configurazione dei componenti

e include valutazioni di coerenza e fattibilità, fondamentali per le fasi successive. Revisioni congiunte assicurano l'aderenza ai criteri tecnici stabiliti.

2.3.1.4) Analisi dei requisiti software

Questa fase traduce le esigenze del sistema$_G$ in requisiti$_G$ specifici per il software$_G$. Si dettagliano le funzionalità attese, le prestazioni, gli aspetti di sicurezza e usabilità, e i vincoli operativi. Ogni requisito viene documentato con specifiche chiare, distinguendo tra requisiti funzionali e non funzionali. Vengono inoltre definite le interfacce e i requisiti per la gestione dei dati e del database$_G$. La tracciabilità con i requisiti di sistema è garantita e revisioni periodiche confermano la coerenza e l'adeguatezza dei requisiti.

2.3.1.5) Progettazione architetturale del software

In questa fase i requisiti$_G$ vengono convertiti in una struttura di alto livello per il software$_G$. Si identificano i moduli, i componenti e le interfacce interne ed esterne, assicurando una corretta distribuzione delle funzionalità. Vengono delineati anche i progetti preliminari per il database$_G$ e la documentazione utente. La revisione degli standard$_G$ di progettazione assicura coerenza e tracciabilità delle scelte tecniche, creando una solida base per la progettazione dettagliata.

2.3.1.6) Progettazione dettagliata del software

Questa fase approfondisce la progettazione definendo algoritmi, strutture dati e logiche operative per ciascun componente del software$_G$. Si producono diagrammi, pseudocodice e documenti tecnici che supportano la codifica, specificando in dettaglio le interfacce tra unità. Vengono aggiornate le specifiche per il database$_G$ e la documentazione$_G$ utente, e si definiscono i requisiti di test. Revisioni interne garantiscono coerenza, completezza e prontezza per la fase di codifica.

2.3.1.7) Codifica e testing del software

In questa fase il codice del software$_G$ viene sviluppato seguendo le specifiche della progettazione dettagliata. Ogni unità viene implementata in modo modulare e accompagnata da procedure di test unitari. I test verificano la conformità del codice ai requisiti$_G$ e al design previsto, con i risultati accuratamente documentati. Si applicano standard di codifica e strumenti di testing per monitorare la qualità e la copertura dei test, aggiornando la documentazione$_G$ in parallelo.

2.3.1.8) Integrazione del software

Questa fase prevede l'unione delle diverse unità e moduli del software$_G$ in un sistema coeso. Viene redatto un piano di integrazione che specifica le procedure, le responsabilità e le tempistiche necessarie per combinare le unità. Durante l'integrazione, vengono eseguiti test specifici per verificare l'interoperabilità e la corretta comunicazione tra i componenti. La documentazione$_G$ viene aggiornata con ogni modifica, e revisioni periodiche confermano che l'aggregato soddisfi i requisiti iniziali.

2.3.1.9) Test di qualifica del software

In questa fase si eseguono test mirati per verificare che ogni requisito$_G$ del software$_G$ sia stato implementato correttamente. Vengono definiti casi di test dettagliati con input, output e criteri di successo, valutando funzionalità, performance, sicurezza e usabilità. I risultati vengono documentati in modo da evidenziare eventuali anomalie e garantire la copertura totale dei requisiti. Strumenti automatizzati e revisioni periodiche assicurano tracciabilità e coerenza durante il testing.

2.3.1.10) Integrazione di sistema

Questa attività combina il software$_G$ con componenti hardware, processi manuali ed altri sistemi esterni, formando un sistema integrato. Viene creato un piano di integrazione di sistema che definisce procedure, responsabilità e criteri di verifica, assicurando che le varie componenti interagiscano correttamente. I test progressivi e la documentazione aggiornata garantiscono che il sistema complessivo soddisfi i requisiti globali, preparandolo per la fase finale di qualifica.

2.3.1.11) Test di qualifica del sistema

In questa fase l'intero sistema, comprendente hardware$_G$ e software$_G$, viene testato in condizioni operative reali. Vengono eseguiti casi di test che simulano scenari d'uso quotidiano, valutando funzionalità, performance e sicurezza a livello globale. I risultati sono documentati per verificare la copertura dei requisiti di sistema e la conformità ai risultati attesi. Revisioni congiunte e strumenti di testing automatizzati supportano la valutazione, garantendo che il sistema sia robusto e pronto per la consegna.

2.3.1.12) Installazione del software

Questa fase riguarda la distribuzione del software$_G$ nell'ambiente di destinazione, seguendo un piano di installazione dettagliato. Vengono identificate le risorse e le informazioni necessarie per configurare il sistema e assicurare il corretto avvio del codice e dei database$_G$. Il processo di installazione viene monitorato e documentato in ogni fase, garantendo il rispetto delle specifiche tecniche e operative. L'assistenza tecnica supporta la configurazione e risolve eventuali problematiche riscontrate durante il processo.

2.3.1.13) Supporto all'accettazione del software

Questa attività finale prevede l'assistenza durante le fasi di revisione e testing di accettazione da parte dell'acquirente. Il programmatore supporta la verifica della conformità del software$_G$ ai requisiti contrattuali, coordinando sessioni di test congiunte e raccogliendo feedback. I risultati dei test e delle revisioni sono accuratamente documentati, garantendo trasparenza e tracciabilità. Viene fornita formazione finale agli utenti e stabilita una baseline definitiva per il prodotto consegnato. Il supporto post-consegna facilita la manutenzione e una transizione operativa fluida.

2.3.2) Strumenti

Il gruppo utilizza una serie di strumenti per supportare lo sviluppo e la progettazione del software$_G$, garantendo efficienza e chiarezza nei processi:

- **Visual Studio Code**: un editor di codice sorgente open source, leggero e altamente personalizzabile. Offre supporto per numerosi linguaggi di programmazione, integrazione con sistemi di controllo versione e una vasta gamma di estensioni per il debugging$_G$ e l'automazione, rendendolo ideale per la scrittura e la revisione del codice.
- **draw.io**: uno strumento online per la creazione di diagrammi$_G$ e schemi, ampiamente utilizzato per realizzare diagrammi$_G$ UML$_G$, flussi di processo e mappe concettuali. Grazie alla sua interfaccia intuitiva, facilita la rappresentazione visiva delle architetture$_G$ e dei processi, migliorando la comunicazione all'interno del team.
- **IntelliJ IDEA**: un ambiente di sviluppo integrato (IDE$_G$) potente e intelligente, particolarmente apprezzato per lo sviluppo in Java$_G$. Fornisce funzionalità avanzate come il completamento

automatico del codice, il refactoring$_G$ e il debugging$_G$, supportando lo sviluppo di applicazioni complesse con elevati standard qualitativi.

2.3.2.1) Utilizzo di Git e della Repository$_G$

Tutto il codice del progetto è contenuto nell'apposita repository GitHub$_G$.

È cura del programmatore tenere sempre la sua repository$_G$ locale aggiornata con quella remota mediante l'utilizzo del comando `git pull` .

Per lo sviluppo di codice, che sia questo a scopo implementativo o di fix, il programmatore dovrà seguire i seguenti passaggi:

1. posizionarsi in un branch locale dedicato, partendo dal **main**, mediante il comando `git checkout -b nuovoBranch`
2. eseguire il commit locale delle modifiche
3. una volta terminata l'attività di codifica associata al branch è necessario fare il pull dello stesso, mediante i comandi:
    1. `git add .` o `git add --all` — per aggiungere i file modificati nell'area di staging
    2. `git commit -m "messaggio"` — per creare un commit con i file aggiunti in staging
    3. `git push --set-upstream origin <nome_branch>` — per caricare il nuovo branch e le modifiche sul repository$_G$ remoto
4. aprire la PR$_G$, assegnando il responsabile$_G$ di progetto come revisore$_G$ in quanto unica persona abilitata, dopo un controllo su quanto fatto, ad effettuare il merge sul main.

Il responsabile di progetto è tenuto a:

1. verificare il lavoro fatto dal programmatore
2. approvare la PR$_G$ mediante l'apposita funzione di GitHub$_G$
3. chiudere la PR$_G$ effettuando il merge sul main ed eliminando il branch oppure richiedere modifiche al programmatore
    - in caso di richiesta modifiche l'iter riparte

3) Processi di Supporto

3.1) Documentazione

Questa sezione tratta le norme per la redazione della documentazione del gruppo, in linea con l'organizzazione del team, allineando lo stile e la gestione delle revisioni.

3.1.1) Modelli di documento

La redazione di tutta la documentazione del gruppo avviene in Typst[G] utilizzando i templates messi a disposizione nell'apposita cartella «templates» della repository[G]
I modelli di documento sono:

- documento
- allegato
- carta intestata

3.1.1.1) Documento

Questo template[G] viene utilizzato per la redazione di tutta la documentazione interna ed esterna.
Nella prima pagina del documento devono essere indicati, oltre a titolo e sottotitolo:

- **autore del documento**
- **tipologia del documento** (Interno[G] o Esterno[G])
- **ultima modifica**
- **stato del documento** (Bozza oppure Approvato)

L'aggiornamento di autore e tipologia del documento è a cura del redattore[G] del documento.
Lo stato viene inserito come «Approvato» solo nei verbali esterni, che hanno bisogno dell'approvazione da parte dell'azienda proponente. Negli altri documenti non viene inserito lo stato.
L'ultima modifica viene aggiornata automaticamente ad ogni modifica della Tabella delle revisioni[G] , prendendo la data dell'ultima revisione[G] come data di ultima modifica.
L'indice si aggiorna automaticamente in base alle sezioni di Typst[G] , per il dettaglio su come suddividere correttamente il documento in sezioni e sottosezioni si rimanda alla documentazione ufficiale di Typst[G] .

Per la gestione della tabella delle revisioni[G] si fa riferimento all'apposita sezione, Sezione 3.1.3, di questo documento.

3.1.1.2) Allegato

Questo template[G] viene utilizzato per la redazione degli allegati ai verbali (interni[G] ed esterni[G]). È compito di chi redige l'allegato indicare, nell'apposita sezione nell'intestazione del documento:

- numero allegato (num progressivo riferito al verbale)
- numero di verbale (esplicitando se interno[G] o esterno[G])
- data del verbale

Il documento di questa tipologia viene inserito nello stesso documento del verbale.

3.1.1.3) Carta intestata

Questo template[G] viene utilizzato per tutte le comunicazioni ufficiali in uscita verso un destinatario esterno[G] .
È compito di chi redige il documento indicare, nell'apposita sezione:

- destinatario del documento
- mezzo di invio del documento
- oggetto del documento

3.1.2) Redazione dei verbali

La modalità di redazione dei verbali interni$_G$ ed esterni$_G$ è la medesima.
Nella prima pagina di contenuto, ovvero la pagina nr. 3, è necessario indicare, in ordine:

- breve sezione, scritta in *italic* con motivo e modalità della convocazione
- ordine del giorno$_G$
- dettagli dell'incontro, con riferimento a:
  - data e ore della convocazione
  - luogo (in presenza oppure online, specificando in questo caso la piattaforma$_G$
  - destinatari dell'incontro
- verbale, specificando:
  - presenze

Dopo le presenze si procede con il riassunto della discussione dei relativi punti dell' OdG$_G$, da riportare in ordine. L'ultima sezione deve sempre essere «**Varie ed eventuali**» indicando, se ci sono state, discussioni di punti extra OdG$_G$ ed orario di fine dell'incontro.

Il verbale deve inoltre contenere, nella relativa sezione del template$_G$:

- una tabella con un riassunto delle decisioni prese. Ogni riga di questa deve contenere il riferimento al punto dell' OdG$_G$, per consentire al lettore di approfondire la sezione di interesse senza dover leggere tutto il documento, l'argomento e la decisione presa.
- una tabella TODO$_G$ con riferimento alle issue$_G$ create relativamente alle decisioni prese. In quest'ultima è necessario indicare ID$_G$ della issue$_G$, assegnatario (se presente, in caso contrario «-»), descrizione del task$_G$.

Alla fine del documento deve essere indicato Luogo e Data, sede del gruppo, e la data della riunione, Verbalizzante$_G$ e Responsabile di Progetto$_G$ e, nel caso di verbale esterno$_G$, firma, per approvazione, di un rappresentante dell'azienda.

3.1.3) Registro delle modifiche e versionamento

Ogni documento, escluso allegati e carta intestata, contiene la tabella delle revisioni con il registro delle modifiche$_G$.

È compito di colui che scrive o modifica un documento:

1. Aggiornare la tabella con:
   - versione del documento, in forma x.y.z (Sezione 3.2)
   - data della modifica
   - descrizione
   - autore della modifica

È compito del verificatore$_G$:

1. verificare la correttezza, sia grammaticale/sintattica che di contenuto, delle modifiche apportate
2. assicurarsi che il numero di versione assegnato alla modifica sia corretto secondo le regole del team

3. inserire il proprio nome nell'apposita colonna della tabella

È compito del responsabile$_G$:
1. verificare la correttezza dei contenuti
2. inserire il proprio nome nell'apposita colonna della tabella, insieme a quello del verificatore

3.1.4) Ciclo di vita

Il ciclo di vita di un documento può essere suddiviso nelle seguenti fasi:

1. **Adattamento o creazione di un template**: si definisce la struttura base del documento, scegliendo o adattando un template conforme agli standard$_G$ stabiliti, garantendo uniformità tra i vari documenti del progetto.
2. **Definizione delle sezioni**: si identificano le sezioni necessarie in base agli obiettivi del documento, assicurandosi che coprano tutti gli aspetti richiesti e siano organizzate in modo logico e coerente.
3. **Definizione degli elaboratori delle sezioni**: ogni sezione viene assegnata a specifici elaboratori$_G$, che hanno il compito di redigerla in base alle linee guida stabilite nel Norme di Progetto$_G$.
4. **Stesura delle sezioni**: gli elaboratori scrivono il contenuto delle sezioni assegnate, seguendo gli standard$_G$ definiti dalle Norme di Progetto$_G$ e utilizzando un linguaggio chiaro e tecnico, se necessario.
5. **Verifica da parte degli elaboratori**: dopo la stesura, gli elaboratori$_G$ eseguono un primo controllo sulla correttezza, coerenza e completezza delle sezioni, assicurando il rispetto delle convenzioni stabilite dalle Norme di Progetto$_G$.
6. **Verifica delle sezioni**: i verificatori$_G$ esaminano il documento per individuare eventuali errori, incongruenze o violazioni delle norme di qualità$_G$, fornendo feedback per eventuali correzioni.
7. **Approvazione da parte del responsabile**: il responsabile$_G$ approva il documento dopo la verifica finale, rilasciando la versione definitiva pronta per la pubblicazione e l'utilizzo ufficiale.

3.1.5) Sistema di composizione tipografica

Per la stesura dei documenti, il gruppo utilizza Typst$_G$, un moderno sistema di composizione tipografica che offre un'alternativa più accessibile rispetto a LaTeX$_G$. Typst$_G$ combina la potenza della formattazione automatizzata con una sintassi più intuitiva e immediata. I principali vantaggi rispetto a LaTeX includono:

- Sintassi più semplice e leggibile, riducendo la curva di apprendimento per nuovi utenti.
- Gestione più intuitiva degli stili e delle strutture, senza necessità di pacchetti complessi.
- Maggiore integrazione con strumenti moderni, facilitando l'uso in ambienti collaborativi.

L'uso di Typst$_G$ quindi, consente una redazione più veloce ed efficiente dei documenti, mantenendo un'elevata qualità tipografica e semplificando la gestione della formattazione.

3.2) Numero di versione

Come indicato il numero di versione è formato da 3 valori: **x.y.z**.
L'incremento del valore **y** avviene ad ogni nuova modifica.

L'incremento del valore **z** avviene ad ogni FIX richiesto da verificatore$_G$ o responsabile$_G$.
L'incremento del valore **x** avviene ad ogni scatto di versione, rispettivamente quindi:

- RTB$_G$ –> *x=1*
- PB$_G$ –> *x=2*

3.3) Standard UML per la stesura di alcuni documenti

All' interno$_G$ dell' analisi dei requisiti$_G$ è ovviamente di fondamentale importanza la presenza dei casi d'uso, descritti come:

- **Descrizione**: una breve descrizione del caso d'uso che identifica chiaramente la funzione che il sistema deve svolgere.
- **Attore**: l'entità che interagisce col sistema, è un'entità esterna su cui non si possono effettuare modifiche.
- **Precondizioni**: le condizioni che definiscono lo stato iniziale del sistema e degli attori prima che l'interazione inizi.
- **Postcondizioni**: le condizioni che descrivono lo stato finale del sistema.
- **Scenario principale**: la sequenza di passi standard che descrive l'interazione principale tra l'attore e il sistema per completare un caso d'uso.
- **Scenari alternativi**: rappresentano dei casi particolari, ovvero quando uno dei passi dello scenario principale non è andato a buon fine ed è dunque necessario specificare come comportarsi in queste circostanze.

Per tutti quei casi d'uso che interagiscono con altri è molto importante inserire il *diagramma del caso d'uso*

3.3.1) Diagramma casi d'uso

I diagrammi dei casi d'uso sono una rappresentazione visiva utilizzata nell'ingegneria del software per descrivere le interazioni tra gli utenti (attori) e un sistema o applicazione. Fanno parte del linguaggio di modellazione UML$_G$ e si concentrano su ciò che il sistema deve fare dal punto di vista dell'utente, piuttosto che sul «come» viene implementato. I diagrammi dei casi d'uso, dunque, aiutano a identificare e documentare i requisiti funzionali del sistema, mostrando cosa deve essere fatto per soddisfare le esigenze degli utenti.

- **Chiarezza nella comunicazione**: Forniscono una rappresentazione semplice e intuitiva, comprensibile sia per i team tecnici che per i non tecnici, come stakeholder e clienti.
- **Base per ulteriori sviluppi**: Servono come punto di partenza per altre attività$_G$ di progettazione e sviluppo, come la definizione dei diagrammi di sequenza, di attività$_G$ o di stato.
- **Riduzione delle ambiguità**: I diagrammi forniscono una visione chiara delle funzionalità$_G$ del sistema, evitando fraintendimenti e malintesi tra i vari membri del team.
- **Strumento di comunicazione universale**: Sono una rappresentazione standardizzata e riconosciuta che facilita la collaborazione tra persone con competenze e background diversi.

Per descrivere un caso d'uso con un suo diagramma dunque, il team utilizza uno standard, indicato nelle sezioni successive

3.3.1.1) Identificare un caso d'uso

Un caso d'uso è una descrizione di una funzionalità$_G$ o servizio specifico offerto da un sistema, visto dal punto di vista dell'utente (attore esterno$_G$). Rappresenta un obbiettivo che un attore cerca di

raggiungere attraverso l'interazione con il sistema. Viene rappresentato tramite un semplice ovale con un nome al suo interno$_G$. La sua nomenclatura è del tipo « UC$_G$ N - Nome caso d'uso»

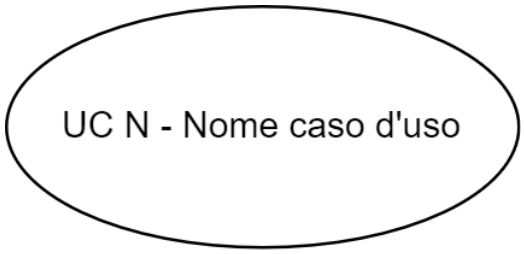


Figura 3: Identificare un caso d'uso

È possibile inoltre identificare un sottocaso d'uso. La nomenclatura è del tipo « UC$_G$ N.n - Nome sottocaso d'uso» dove si intende che è è il sottocaso n del caso d'uso N

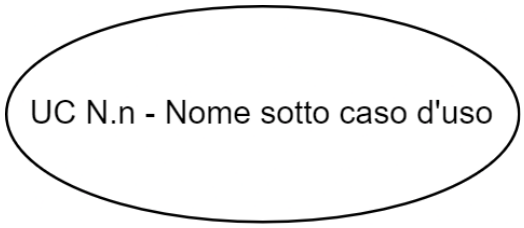


Figura 4: Identificare un sottocaso d'uso

3.3.1.2) Identificare un attore

Un attore è un'entità esterna che interagisce con un sistema o applicazione per raggiungere un obiettivo specifico. L'attore rappresenta un «omino stilizzato» che utilizza o interagisce con il sistema modellato, ma non fa parte del sistema stesso.

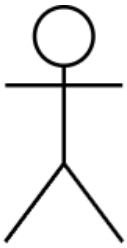


Figura 5: Identificare un attore

3.3.1.3) Identificare la webapp (sistema)

Il sistema, nel nostro caso la webapp, indica la «zona» in cui i casi (o sottocasi) d'uso vengono inseriti. È rappresentato da un semplice rettangolo con all' interno$_G$ i casi d'uso e all' esterno$_G$ l'attore

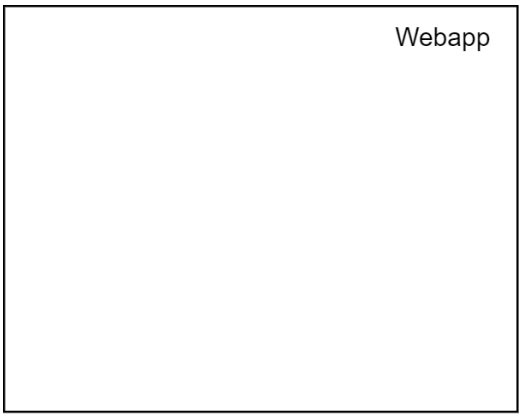


Figura 6: Identificare la webapp (sistema)

3.3.1.4) Identificare le relazioni

Le relazioni nei diagrammi dei casi d'uso descrivono come gli elementi del sistema interagiscono tra loro. In particolare, rappresentano le connessioni tra attori e casi d'uso, oppure tra diversi casi d'uso. Servono a chiarire il comportamento del sistema, mostrando le dipendenze, le collaborazioni e le estensioni delle funzionalità[G] . Ci sono vari tipi di relazioni: associazione, inclusione, estensione e generalizzazione.

3.3.1.4.1) Associazione

Collega un attore a un caso d'uso, indicando che l'attore interagisce con quel caso. È rappresentato da una semplice linea

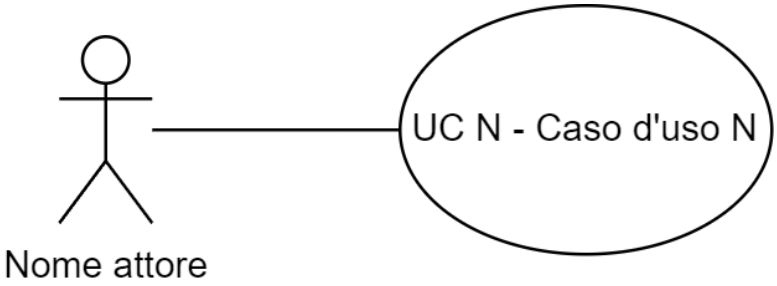


Figura 7: Identificare la relazione associazione

3.3.1.4.2) Inclusione

Indica che un caso d'uso include un altro caso come parte del suo flusso principale e si utilizza per evitare ripetizioni di azioni comuni a più casi d'uso. Viene rappresentata da una freccia tratteggiata con etichetta «include».

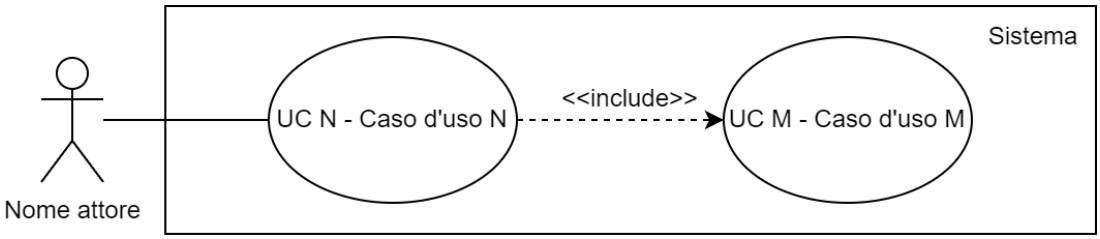


Figura 8: Identificare la relazione inclusione

3.3.1.4.3) Estensione (e condizioni)

Indica che un caso d'uso può estendere un altro caso d'uso aggiungendo comportamenti opzionali o condizionali. In sostanza quindi aumenta le funzionalità[G] di uno use case. È rappresentata da una freccia tratteggiata con etichetta «extend».
Attenzione: la freccia va dal caso da cui si vuole estendere verso il caso d'uso che estende.

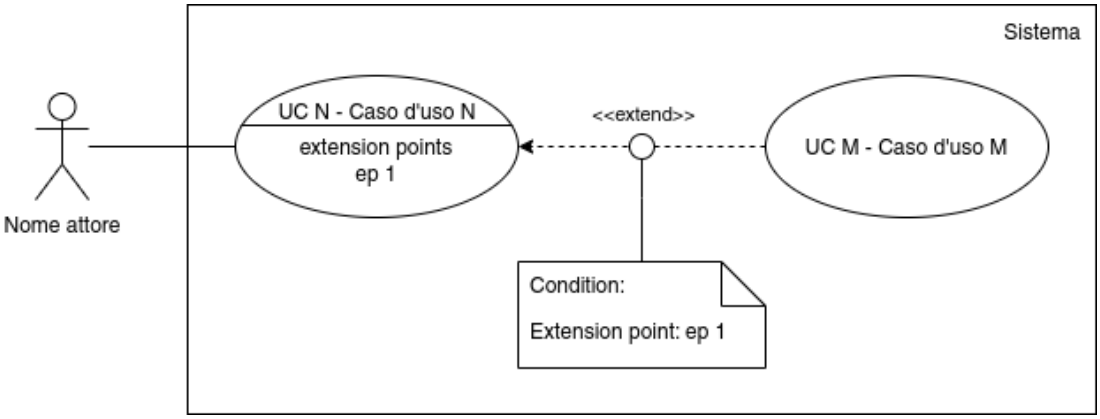


Figura 9: Identificare la relazione estensione

In questo esempio, il caso d'uso N estende il caso d'uso M aggiungendo delle funzionalità[G] specifiche, vincolate dalla *condizione*. L'*extension point* rappresenta il punto in cui il caso d'uso da estendere può essere esteso, a seconda del verificarsi della *condizione* specificata. La *condizione* viene indicata collegando un commento alla freccia di estensione, contenente la descrizione della *condizione* stessa e l'*extension point* corrispondente. È importante segnalare l'*extension point* anche nel caso d'uso esteso.

3.3.1.4.4) Generalizzazione

Lo scopo principale è di aggiungere o modificare le caratteristiche di base. Indica quindi un rapporto gerarchico, dove un attore o un caso d'uso più specifico eredita caratteristiche da uno più generico. Si utilizza per rappresentare specializzazioni di ruoli o di comportamenti ed è rappresentata da una freccia con linea continua e punta vuota (non tratteggiata).

Esistono quindi due tipi di generalizzazioni: generalizzazione tra attori e generalizzazioni tra use case

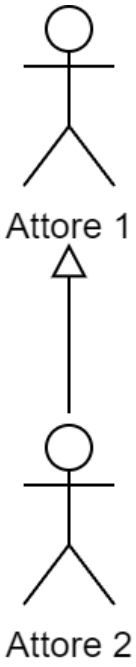


Figura 10: Identificare la relazione generalizzazione tra attori

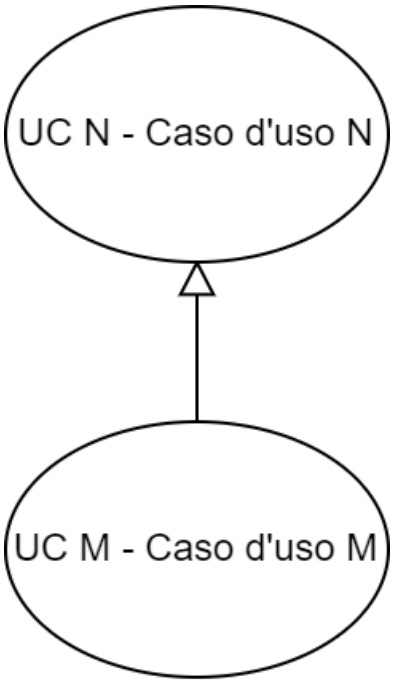


Figura 11: Identificare la relazione generalizzazione tra casi d'uso

Pertanto, i figli possono aggiungere delle funzionalità[G] rispetto ai padri oppure modificarne il comportamento

3.3.2) Diagramma delle classi

Nei futuri documenti, durante le fasi di progettazione, sarà necessario rappresentare graficamente non più solo quello che l'utente desidera poter fare (interazioni con il sistema) ma anche come lo si

andrà ad implementare. A questo scopo è quindi fondamentale impostare delle regole UML$_G$ per la rappresentazione delle classi tramite diagrammi. Le funzionalità$_G$ principali sono:

- **Modellare la struttura del sistema**: Forniscono una visione dettagliata di come le entità del sistema (classi) sono definite e come interagiscono tra loro.
- **Progettazione e Documentazione**$_G$: Sono utili durante la fase di progettazione per definire l'architettura del software e durante lo sviluppo per documentare il sistema.
- **Supporto per il Codice**: Possono essere utilizzati durante la fase di sviluppo per scrivere il codice o per comprendere e modificare un sistema esistente.
- **Facilitare la Comunicazione**: Rappresentano un linguaggio comune tra sviluppatori, analisti, progettisti e stakeholder.

Come per i diagrammi UML$_G$ per i casi d'uso, il team utilizzerà degli «standard» per la scrittura dei diagrammi delle classi, descritti nelle sezioni successive

3.3.2.1) Identificare una classe

Una classe in UML$_G$ è rappresentata come un rettangolo diviso in tre sezioni:

- **Nome della classe**: La prima sezione contiene il nome della classe, scritto in grassetto e centrato (in corsivo se la classe è astratta).
- **Attributi**: La seconda sezione elenca le proprietà della classe, indicando il tipo di dati e la visibilità (es. privato o pubblico).
  La definizione è:
  Visibilità nome : tipo [molteplicità] = default {proprietà aggiuntive}
- **Metodi**: La terza sezione elenca i metodi o le operazioni che la classe può eseguire, specificandone i parametri e il tipo di ritorno.

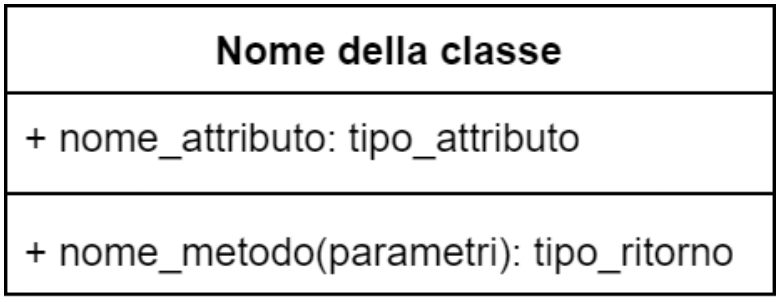


Figura 12: Identificare una classe

Il simbolo che c'è prima di un'attributo è detto *visibilità* e può essere di 4 tipi:

- + indica che la visibilità è pubblica
- - indica che la visibilità è privata
- # indica che la visibilità è protetta
- ~ indica che la visibilità è di package

Le *proprietà aggiuntive* possono essere:

- *Ordered* : Per array o vettori
- *Unordered*: Per gli insiemi

3.3.2.2) Identificare le relazioni

Come per quanto riguarda gli UML$_G$ per i casi d'uso, le relazioni tra classi sono molto importanti per indicare come si «relazionano» tra loro.

3.3.2.2.1) Associazione

Identifica che un'istanza di una classe è legata a una o più istanze di un'altra classe. Viene identificata da una semplice linea continua orientata.

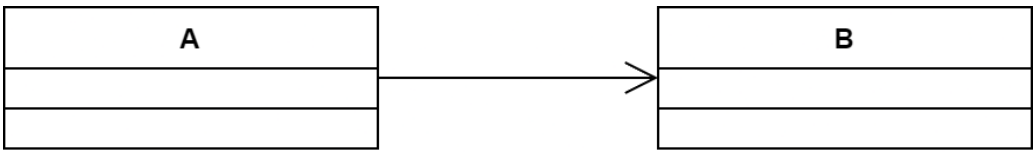


Figura 13: Identificare la relazione associazione tra classi

Alla fine della freccia è possibile indicare la *molteplicità*:

- **1** (uno a uno): Un oggetto di una classe è associato a un solo oggetto dell'altra.
- **0..1** (zero o uno): Un oggetto di una classe può essere associato a nessun oggetto o 1 oggetto dell'altra classe.
- **0..*** (zero o più): Un oggetto di una classe può essere associato a nessun oggetto o a più oggetti dell'altra classe.
- **1..*** (uno o più): Almeno un oggetto di una classe è associato a uno o più oggetti dell'altra.
- **n** (es. 3): Un numero specifico di associazioni.

3.3.2.2.2) Aggregazione

Rappresenta un legame «parte di» tra due classi. In questa relazione, una classe (il «tutto») è composta da una o più istanze di un'altra classe (le «parti»), ma le «parti» possono esistere indipendentemente dal «tutto». Dunque un oggetto è composto da altri oggetti, ma le parti possono avere una vita indipendente e non sono distrutte quando il tutto viene eliminato. Viene indicato con una linea e un rombo vuoto vicino alla classe che è parte di un'altra classe

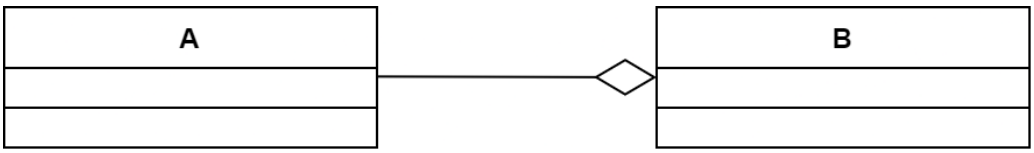


Figura 14: Identificare la relazione aggregazione tra classi

Quindi in questo esempio B è parte di A.

3.3.2.2.3) Composizione

È una relazione simile all'aggregazione, con il sostanziale cambiamento che una classe fa parte di un'altra classe, ma non può esistere indipendentemente, ma solo se fa parte dell'altra classe. Viene rappresentato tramite una linea con un rombo pieno alla fine.

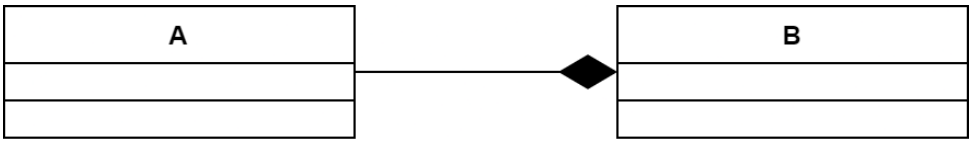


Figura 15: Identificare la relazione composizione tra classi

Quindi in questo esempio B è parte della classe A ma B non può esistere indipendentemente, ma solo come parte di A

3.3.2.2.4) Generalizzazione

A generalizza B, se ogni oggetto di B è anche un oggetto di A
Il concetto è quello dell'ereditarietà della programmazione ad oggetti dove ci sono classi che ereditano da altre (chiamate classi figlie o sottoclassi). Viene identificata da una freccia con la punta vuota.

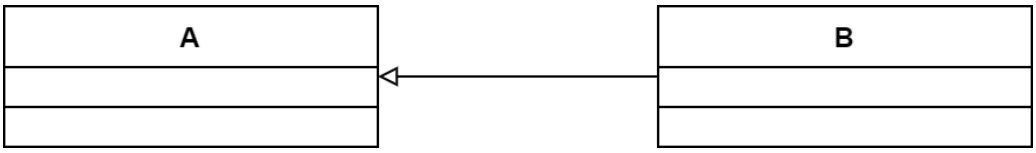


Figura 16: Identificare la relazione generalizzazione tra classi

Quindi in questo esempio B è figlia di A (superclasse)

3.3.2.2.5) Dipendenza

La *definizione* è: *Si ha dipendenza tra due elementi di un diagramma se la modifica alla definizione del primo (fornitore) può cambiare la definizione del secondo (client).* **È importante che le dipendenze siano minimizzate (*loose coupling*)**.
Indica che una classe utilizza l'altra per svolgere una funzione specifica o per accedere a un servizio. In altre parole, una classe dipende da un'altra quando un cambiamento nella classe dipendente potrebbe influenzare la classe che dipende da essa. È quindi una relazione «debole» grazie al suo basso grado di accoppiamento. È identificato da una freccia tratteggiata orientata.

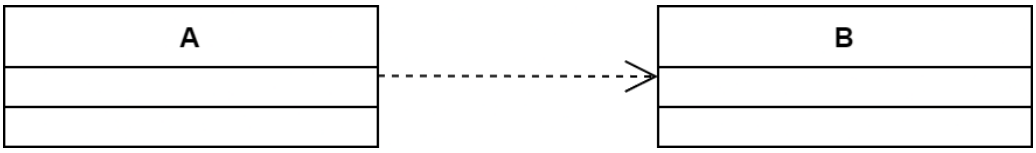


Figura 17: Identificare la relazione dipendenza tra classi

Quindi un cambiamento in B *potrebbe* influenzare A, ma un cambiamento in A *non influenza* B.

3.3.2.3) Classi di associazione

Aggiungono attributi e operazioni alle associazioni. Sono utilizzate per rappresentare informazioni aggiuntive o comportamenti specifici che appartengono a una relazione tra due (o più) classi, ma che non sono strettamente parte di nessuna delle due classi connesse. Per la rappresentazione è una classe (rettangolo) collegata alla linea di associazione tramite una linea tratteggiata.

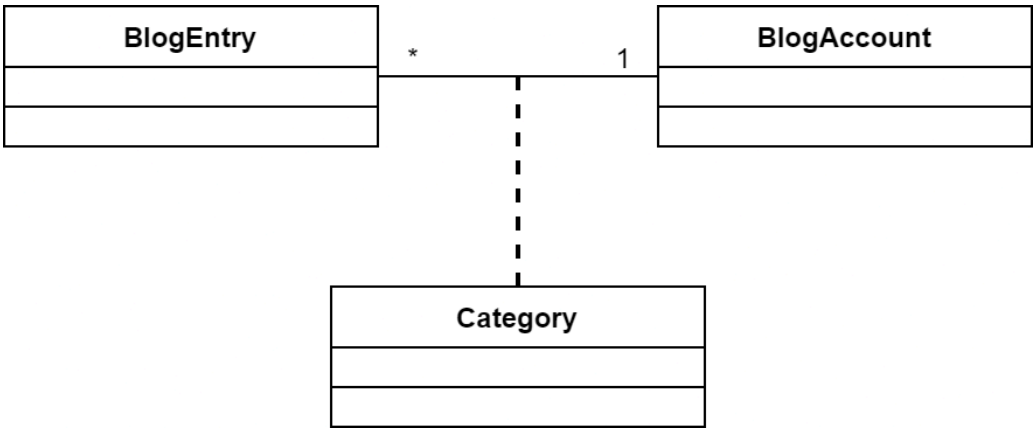


Figura 18: Identificare le classi di associazione

3.3.2.4) Interfacce

L'interfaccia è una classe priva di implementazione. Una classe *realizza* un'interfaccia se ne implementa le operazioni. Con UML$_G$ 2.x viene identificata come un cerchio (*«Ball» notation*)

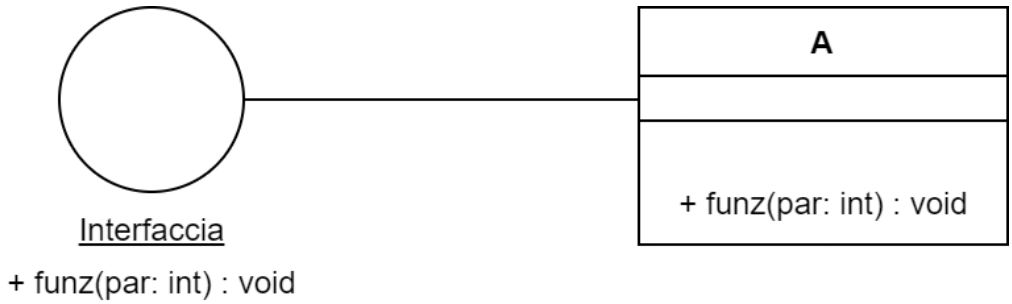


Figura 19: Identificare le classi di associazione

In questo caso la classe A implementa la funzione «funz» dell'interfaccia «Interfaccia» che non ha la sua implementazione

È possibile inoltre dichiarare il fatto che una classe ha la necessità di «collegarsi» con un'interfaccia per poter accedere ad alcune funzionalità$_G$. Ad esempio:

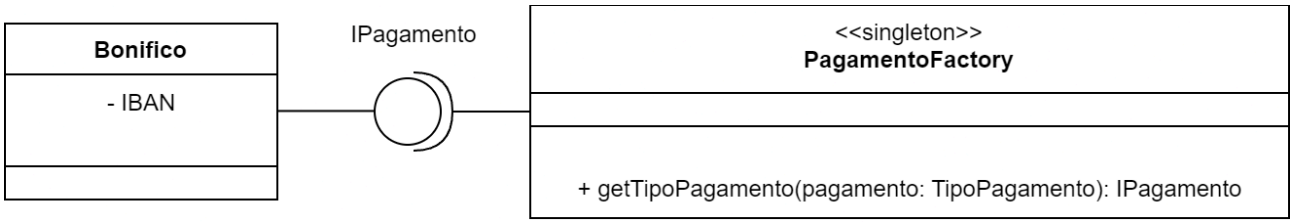


Figura 20: Identificare le classi di associazione

Questo significa che il *singleton* PagamentoFactory ha bisogno (require) dell'interfaccia *IPagamento* per «accedere» a *Bonifico*.

3.4) Acronimi ed abbreviazioni

Nella documentazione prodotta dal gruppo vengono utilizzati, vista la ripetizione dei termini, i seguenti acronimi e le seguenti abbreviazioni:

Acronimo	Parola
AdR$_G$	Analisi dei Requisiti$_G$
UC$_G$	Use Case
NdP$_G$	Norme di Progetto
PdP$_G$	Piano di Progetto
PdQ$_G$	Piano di Qualifica$_G$
PoC$_G$	Proof of Concept
RTB$_G$	Requirements and Technology Baseline
PB$_G$	Product Baseline
MVP	Minimum Viable Product

Abbreviazione	Parola
G	Glossario

3.5) Verifica e Revisione della documentazione

Per il processo di verifica ed approvazione dei documenti il team utilizza le Pull Request$_G$ di GitHub$_G$.

Vengono riportati nelle sezioni successive i passaggi che ogni figura coinvolta in questo processo dovrà fare.

3.5.1) Processo per la verifica della documentazione

Questa sezione presenta tutte le istruzioni che vengono applicate, dalla creazione/modifica del file fino alla sua verifica, per garantire la qualità del documento.

3.5.1.1) Redattore

1. `git checkout sources` — per spostarsi sul branch di lavoro
2. `git pull` — per scaricare le ultime modifiche
3. `git checkout -B <nome_branch>` — per creare un nuovo branch di lavoro, partendo dal branch di lavoro sources
4. Crea dei file o modifica i file esistenti
5. `git add .` o `git add --all` — per aggiungere i file modificati nell'area di staging
6. `git commit -m "messaggio"` — per creare un commit con i file aggiunti in staging
7. `git push --set-upstream origin <nome_branch>` — per caricare il nuovo branch e le modifiche sul repository$_G$ remoto
8. Aprire la Pull Request$_G$
   - La Pull Request$_G$ può essere aperta tramite un pulsante «Create Pull Request$_G$ « presente nella pagina iniziale del repository$_G$
   - La Pull Request$_G$ può essere aperta andando nella pagina «Pull Requests», impostando «nome_branch» come branch sorgente e «sources» come branch di destinazione. Premere successivamente il pulsante «Create Pull Request$_G$ «

- **ATTENZIONE**. Impostare il merge al branch *sources* (viene selezionato in automatico se è stato creato il nuovo branch a partire dal branch *sources*). È molto importante fare sempre attenzione a questo punto, per non incorrere a problemi di merge.

9. Una volta creata, si assegna il verificatore$_G$ nella sezione «Reviewers» a destra della pagina della Pull Request$_G$ senza dimenticare d'inserire anche il responsabile$_G$, le labels, la board sotto la voce project e la milestone$_G$ se presenti.
10. Collega la/le issue/issues alla Pull Request$_G$ nella sezione «Development» a destra della pagina per la modifica della stessa. Questo permette di chiudere tutte le issue$_G$ associate una volta che la Pull Request$_G$ è stata approvata.
    - **ATTENZIONE**. L'impostazione delle issue$_G$ va effettuata **DOPO** la creazione della Pull Request$_G$ e non prima. Questo serve per garantire che venga aggiunto il messaggio del link tra issue$_G$ e Pull Request$_G$.

Il merge verso il branch *sources* verrà effettuata dal responsabile$_G$ solo dopo la modifica/verifica del documento.

3.5.1.2) Verificatore - Responsabile

Questa sezione presenta tutte le istruzioni a cui attenersi, dal momento in cui il documento è stato modificato fino alla sua verifica.

1. `git pull` — per scaricare le ultime modifiche
2. `git checkout <nome_branch>` — per spostarsi sul branch dove ci sono le modifiche da verificare
3. Controlla i documenti che sono stati modificati
    - Se ci sono errori di battitura o sintattici, corregge il documento in locale procedendo poi con i commit

    1. `git add .` o `git add --all` — per aggiungere i file modificati nell'area di staging
    2. `git commit -m "messaggio"` — per creare un commit con i file aggiunti in staging
    3. `git push` — per caricare le modifiche sul branch
4. Decidere se approvare o meno la Pull Request$_G$
    - Se si decide di non approvarla per mancanza di informazioni importanti, si dovrà rifiutare la Pull Request$_G$ e indicare i motivi del rifiuto:
        1. Premere su «Add your review» in alto a destra
        2. Premere su «Review changes» e selezionare «Request changes», scrivendo i motivi del rifiuto
        3. Premere su «Submit review»
        4. Attendere che il relatore$_G$ apporti le modifiche richieste
    - Se si decide di approvare la Pull Request$_G$, procedere con i seguenti passaggi per il merge:
        1. Premere su «Add your review» in alto a destra
        2. Premere su «Review changes» e selezionare «Approve»
        3. Premere su «Submit review»

Tutte le istruzioni sopra descritte sono valide anche per il responsabile$_G$ che dovrà inoltre seguire le indicazioni riportate di seguito (quest'ultime devono essere ignorate dal verificatore$_G$).

1. Premere su «Merge pull request$_G$ « e successivamente su «Confirm merge»

2. Una volta effettuato il merge, comparirà un bottone «Delete branch» che permette di eliminare il ramo di lavoro. Questo passaggio è fondamentale per mantenere pulita la repository$_G$ e non avere branch inutilizzati.

3.5.2) Analisi statica

L' analisi statica$_G$ è una forma di verifica del software$_G$ che non richiede l'esecuzione del programma. Invece di eseguire il codice, lo analizza insieme alla documentazione per verificare la conformità a regole predefinite, l'assenza di potenziali difetti e la presenza di proprietà desiderate. Questa tecnica può essere applicata a tutti i prodotti del processo di sviluppo software$_G$, non solo al codice eseguibile, e viene utilizzata anche nella validazione.

L'analisi statica può essere eseguita attraverso diversi metodi:

- Per prodotti più semplici, si possono utilizzare metodi di lettura, come il desk check, il walkthrough e l'inspection. La loro efficacia dipende dall'esperienza dei verificatori$_G$ o dalla precisione di strumenti automatizzati.
- Per analisi più approfondite e in contesti in cui la prova empirica è costosa, si adottano metodi formali, come quelli algebrici, basati sulla dimostrazione assistita di proprietà.

Esistono diverse tipologie di analisi statica del codice, tra cui:

- **Analisi del flusso di controllo**: verifica che l'esecuzione avvenga nella sequenza specificata e che il codice sia ben strutturato, individuando anche porzioni non raggiungibili.
- **Analisi del flusso dei dati**: studia le modalità di accesso alle variabili (lettura, scrittura) per rilevare anomalie come scritture successive senza letture intermedie o letture che precedono le scritture. Mira inoltre ad accertare l'assenza di variabili globali$_G$.
- **Analisi dei limiti**: verifica che i valori del programma rimangano sempre entro i limiti definiti dal loro tipo di dato$_G$, controllando problemi come overflow$_G$ e underflow$_G$.
- **Analisi dell'uso dello stack**: studia le dipendenze e le dinamiche di crescita dello stack durante l'esecuzione del programma.
- Altre forme di analisi statica$_G$ includono:
  - **Analisi del flusso dell'informazione**
  - **Esecuzione simbolica**
  - **Verifica formale del codice**
  - **Analisi del comportamento temporale**
  - **Analisi di interferenza**
  - **Analisi del codice oggetto**

L' analisi statica$_G$ è una fase essenziale del processo di verifica e validazione$_G$ e precede, oltre a integrare, l' analisi dinamica$_G$ (i test). Scrivere codice che faciliti la verifica, evitando funzionalità poco chiare, è fondamentale per una buona analisi statica$_G$.

3.5.3) Analisi dinamica

L' analisi dinamica$_G$ è una forma di verifica del software che implica l'esecuzione del codice per osservare il suo comportamento in un ambiente controllato. A differenza dell' analisi statica$_G$, che esamina il codice senza eseguirlo, l' analisi dinamica$_G$ si basa sull'esecuzione di oggetti di prova$_G$, ovvero programmi eseguibili che includono la porzione di codice sotto esame. Ogni prova (test) consiste nell'esecuzione di tale programma con specifici casi di prova$_G$. Un caso di prova$_G$ tipicamente definisce l'oggetto di prova, i valori di ingresso, l'uscita attesa, l'ambiente di esecuzione e lo

stato iniziale, nonché i passi di esecuzione. L'obiettivo principale dell' analisi dinamica$_G$ è rilevare la presenza di difetti nel software$_G$ osservando discrepanze tra il comportamento effettivo e quello atteso. Le prove dovrebbero essere ripetibili e, idealmente, automatizzate per garantire coerenza e facilitare la regressione$_G$.

Un elemento cruciale nell' analisi dinamica$_G$ è l' oracolo$_G$, un meccanismo per determinare a priori i risultati attesi di un test e convalidare i risultati ottenuti. L' analisi dinamica$_G$ è complementare all' analisi statica$_G$ e contribuisce in modo significativo alla misurazione della qualità del prodotto,comprendendo diverse categorie di test che focalizzano la verifica su differenti livelli e aspetti del software$_G$.

3.5.3.1) Test di unità (TU)

Il test di unità$_G$ si concentra sulla verifica della più piccola unità di software utilmente testabile come entità singola, che tipicamente corrisponde a una singola procedura, una classe o un piccolo aggregato coeso. L'obiettivo è accertare la correttezza del codice «as implemented». I test di unità$_G$ possono essere di due tipi principali:

- **Test funzionali$_G$ (black-box)**: Si basano unicamente sulla specifica degli ingressi e delle uscite dell'unità sotto test, senza considerare la sua logica interna. Si utilizzano dati di ingresso che corrispondono a specifici esiti e le classi di equivalenza (insiemi di dati di ingresso che producono lo stesso comportamento funzionale) formano singoli casi di prova$_G$. I TU funzionali contribuiscono alla requirements coverage$_G$, ovvero alla percentuale di requisiti funzionali soddisfatti dal prodotto.
- **Test strutturali$_G$ (white-box)**: Verificano la logica interna del codice dell'unità, perseguendo un alto grado di structural coverage$_G$. Un singolo caso di prova$_G$ attiva un singolo cammino di esecuzione nell'unità. Le dimensioni della structural coverage$_G$ includono Statement Coverage$_G$ (ogni comando eseguito almeno una volta), Branch Coverage$_G$ (ogni ramo del flusso di controllo attraversato almeno una volta) e Decision/Condition Coverage$_G$ (ogni condizione di ogni decisione assume entrambi i valori di verità almeno una volta).

Per eseguire i test di unità$_G$, spesso si utilizzano driver$_G$, componenti fittizi che pilotano il test fornendo gli ingressi, e stub$_G$, componenti fittizi che simulano le parti del sistema dipendenti dall'unità sotto test.

3.5.3.2) Test di integrazione (TI)

Il test di integrazione$_G$ si applica alle componenti individuate nel design architetturale$_G$ e mira a verificare la loro interazione e collaborazione una volta integrate. L'obiettivo è rilevare difetti di progettazione architetturale o bassa qualità dei test di unità$_G$, assicurando che i dati scambiati tra le interfacce siano conformi alla specifica e che i flussi di controllo specificati siano corretti. L'integrazione può avvenire in modo incrementale, costruendo e verificando il sistema passo dopo passo. Questa strategia può comportare l'uso di molti stub$_G$, specialmente in approcci top-down che integrano prima le funzionalità di più alto livello.

3.5.3.3) Test di sistema (TS)

Il test di sistema$_G$ verifica come l'esecuzione del sistema nel suo complesso soddisfi i requisiti software definiti nell' Analisi dei Requisiti$_G$ (AdR). Completa la misura della requirements coverage$_G$ iniziata con i test di unità$_G$ funzionali. Il test di sistema$_G$ è tipicamente funzionale (black-box)

e non richiede conoscenza della logica interna del software. Inizia al completamento del test di integrazione$_G$ e precede il collaudo$_G$.

3.5.3.4) Test di accettazione (Collaudo)

Il test di accettazione$_G$, o collaudo$_G$, accerta il soddisfacimento dei requisiti utente (definiti nel capitolato$_G$) alla presenza del committente. È un'attività formale che, in caso di esito positivo, porta al rilascio finale del prodotto.

3.5.3.5) Test di regressione

Il test di regressione$_G$ ha lo scopo di accertare che le modifiche apportate al software$_G$ per aggiunte, correzioni o rimozioni non pregiudichino le funzionalità già verificate. Il rischio di regressione aumenta con l'aumentare dell' accoppiamento$_G$ e al diminuire dell' incapsulamento$_G$ tra i componenti. Il test di regressione$_G$ comprende la ripetizione selettiva di test di unità$_G$, di integrazione$_G$ e di sistema$_G$ necessari per verificare che la modifica di una parte del sistema$_G$ non causi errori in quella parte o in altre parti correlate.

3.6) Comunicazione interna

La comunicazione interna del gruppo, fondamentale per lo svolgimento del progetto e allineamento dei task$_G$, si divide in due categorie:

- Comunicazione **sincrona**
- Comunicazione **asincrona**

3.6.1) Comunicazione sincrona

3.6.1.1) Periodi di due settimane

Il team, per allinearsi, si riunisce online il **martedì pomeriggio** della settimana in cui non è previsto l'incontro con l'azienda proponente$_G$. In questa riunione ogni membro del team relaziona quanto fatto nel periodo$_G$ in corso, evidenziando eventuali criticità o fattori di rallentamento nello sviluppo dei task$_G$. Viene poi fatta una mini retrospettiva$_G$ complessiva che consente di assumere eventuali decisioni per la prevenzione o risoluzione di problematiche non previste. Durante questo incontro viene aggiornata la project board$_G$ con le nuove issue$_G$ assegnate ai membri. Al termine di ogni incontro sarà cura dell' amministratore$_G$ redigere apposito verbale interno$_G$.
È a cura del responsabile$_G$ di progetto valutare, concordando con il gruppo, eventuali riunioni di allineamento aggiuntive.
Ogni incontro dovrà, in ogni caso, essere preceduto da convocazione mediante i canali di messaggistica del team.

3.6.1.2) Periodi di una settimana

Il team si incontra una volta a settimana a conclusione del periodo corrente, utile per una retrospettiva$_G$, e per iniziare il periodo successivo, con relativa suddivisione dei ruoli e dei compiti. Ogni incontro prevede che venga redatto il relativo verbale interno$_G$, con successivo aggiornamento della repo con le nuove issue. Ogni incontro dovrà, in ogni caso, essere preceduto da convocazione mediante i canali di messaggistica del team.

3.6.1.3) Strumenti

Le riunioni online del team avvengono attraverso la piattaforma$_G$ **Discord**$_G$.
Le convocazioni avvengono invece, nei seguenti canali di messaggistica :

- **Discord**$_G$, canale di comunicazione ufficiale del team
- **Telegram**$_G$, canale di comunicazione informale del team

3.6.2) Comunicazione asincrona

La comunicazione asincrona avviene sia tra tutto il team che tra i singoli componenti, attraverso i canali di comunicazione del gruppo e le piattaforme di messaggistica.
Questo tipo di comunicazione risulta fondamentale per consentire il corretto proseguimento dei task$_G$ senza il vincolo delle sole riunione.

3.6.2.1) Strumenti

Le comunicazioni tra tutti i membri del gruppo avvengono nei canali di messaggistica messi a disposizione, ovvero:

- **Discord**$_G$, canale di comunicazione ufficiale del team
- **Telegram**$_G$, canale di comunicazione informale del team

Le comunicazioni interne tra i membri del gruppo, invece, possono avvenire in modalità di messaggistica o riunione online scegliendo tra le piattaforme gratuite presenti in rete.

3.7) Validazione

L' attività$_G$ di validazione$_G$ viene svolta dimostrando che il prodotto software risponda ai requisiti degli utenti finali attraverso test, prove e altri metodi oggettivi. L'obiettivo dell' attività$_G$ di validazione$_G$ è anche, attraverso l'interazione diretta con il committente, di dimostrare che il prodotto software rispetti tutti i requisiti concordati e che esso funzioni correttamente avendo quindi un prodotto software pronto al rilascio$_G$. Dunque potremmo definire questa attività$_G$ come il processo di accertamento che il prodotto software soddisfi i requisiti specificati e che sia conforme all'uso previsto, come spuntare una checklist di controllo.

3.8) Qualità

3.8.1) Gestione della qualità

L' attività$_G$ di gestione della qualità è un processo$_G$ ampiamente descritto nel documento PdQ$_G$.
La gestione della qualità è essenziale per garantire che il software, la documentazione e le attività$_G$ di sviluppo rispettino gli standard e soddisfino i requisiti definiti. Questo processo si basa su principi strutturati che permettono di monitorare e migliorare costantemente la qualità del progetto. Per ottenere risultati efficaci inoltre, vengono implementati metodi di verifica$_G$ e validazione$_G$, con il coinvolgimento attivo di tutti i membri del team e il supporto di strumenti specifici.

3.8.2) Standard ISO 9000:2000

Lo standard ISO 9000:2000$_G$ fornisce una base solida per la gestione della qualità attraverso un insieme di principi che consentono alle organizzazioni di migliorare continuamente le loro prestazioni. Questi principi sono:

- **Centralità del cliente**: la soddisfazione del cliente è un obiettivo prioritario e la qualità deve essere orientata alle sue esigenze.
- **Leadership**: una chiara direzione strategica e una guida efficace sono fondamentali per garantire il successo del progetto.
- **Partecipazione attiva del team**: il coinvolgimento e la responsabilizzazione di tutti i membri favoriscono il raggiungimento degli obiettivi.

- **Approccio per processi**: ogni attività[G] deve essere gestita come parte di un sistema interconnesso di processi.
- **Miglioramento continuo**: l'evoluzione costante dei metodi di lavoro e delle strategie di sviluppo è essenziale per mantenere alti standard qualitativi.
- **Decisioni basate sui dati**: le scelte devono essere supportate da analisi oggettive e dati concreti.
- **Gestione dei rapporti con i fornitori**: stabilire relazioni di fiducia con fornitori e stakeholder migliora la qualità complessiva.

Tali principi vengono adottati per strutturare un sistema di gestione della qualità che favorisca la coerenza, l'efficienza e l'affidabilità del progetto.

3.8.3) Gestione del cambiamento

Nel contesto di un progetto software, la gestione del cambiamento rappresenta un elemento cruciale per garantire un'evoluzione efficace del processo[G] di sviluppo. Il cambiamento può essere percepito come una sfida o come un'opportunità a seconda dell'approccio adottato dal team. Un cambiamento reattivo si verifica quando il team si adatta forzatamente a nuove situazioni senza una pianificazione adeguata, generando possibili inefficienze. Al contrario, un approccio proattivo permette di anticipare le necessità di modifica, valutandone i benefici e pianificandone l'implementazione in modo strutturato.
Gli elementi chiave per una gestione efficace del cambiamento includono:

- **Identificazione delle necessità di cambiamento**: analisi delle aree che necessitano miglioramenti per ottimizzare il processo[G] di sviluppo.
- **Comunicazione chiara**: coinvolgimento del team e degli stakeholder per garantire una transizione graduale e condivisa.
- **Pianificazione e attuazione**: definizione di strategie operative e assegnazione di responsabilità per gestire il cambiamento con il minimo impatto sul progetto.
- **Monitoraggio e adattamento**: valutazione continua degli effetti del cambiamento e eventuali correzioni per migliorarne l'efficacia.

3.9) Configurazione

L' attività[G] di gestione della configurazione è un processo che norma l'identificazione, organizzazione e controllo delle modifiche agli "artefatti" durante il loro ciclo di vita[G] .

3.9.1) Repository

Le repository[G] del team archi7echs sono:

- **archi7echs-team.github.io**: repository[G] usata per il versionamento, gestione e sviluppo dei file sorgente relativi alla documentazione nel branch sources, per generare la pagina web viene usato il branch website mentre i documenti compilati si possono trovare nel branch master
  Riferimento: https://github.com/Archi7echs-Team/archi7echs-team.github.io - *Ultimo accesso al link 11/03/2025*
- **PoC**[G] : repository[G] usata per il PoC[G]
  Riferimento: https://github.com/Archi7echs-Team/PoC - *Ultimo accesso al link 11/03/2025*
- **MVP**[G] : repository[G] usata per il prodotto finale

Riferimento: https://github.com/Archi7echs-Team/MVP - *Ultimo accesso al link 11/03/2025*

3.9.2) Struttura repository documentazione

La struttura del branch master è (le directory sono segnate in grassetto):

- **documents**
  - **Candidatura**
    - **Verbali**
    - Analisi capitolati
    - Lettera Candidatura
    - Preventivo
  - **RTB**G
    - **Esterni**G
      - **Verbali**
      - Analisi dei requisitiG
      - Piano di Progetto
      - Piano di QualificaG
      - Lettera di Presentazione RTB
    - **Interni**G
      - **Verbali**
      - Glossario
      - Norme di Progetto
  - **PB**G
    - **Esterni**G
      - **Verbali**
      - Analisi dei requisitiG
      - Piano di ProgettoG
      - Piano di QualificaG
    - **Interni**G
      - **Verbali**
      - Glossario
      - Norme di Progetto

3.9.3) Sincronizzazione

La sincronizzazione delle attivitàG viene gestita attraverso repositoryG condivise su GitHubG , garantendo un flusso di lavoro organizzato e tracciabile. Ogni compito assegnato viene monitorato tramite specifiche issueG , che permettono di mantenere una chiara visione dello stato di avanzamento del progetto. Durante lo svolgimento delle attivitàG , ogni membro del team lavora su una versione separata del codice, evitando conflitti e sovrapposizioni. Questo metodo consente di operare parallelamente su più task, ottimizzando la produttività e riducendo il rischio di interferenze tra le modifiche effettuate da diversi sviluppatori.
Le modifiche apportate vengono periodicamente integrate nel flusso principale attraverso pull requestG , che permettono una revisione strutturata e garantiscono che solo codice verificato venga incorporato nel progetto.

3.10) Risoluzione dei problemi

Durante lo sviluppo di un progetto complesso, possono emergere ostacoli di varia natura che, se non affrontati tempestivamente, rischiano di compromettere il corretto avanzamento delle attivitàG .

Il processo[G] di risoluzione dei problemi ha quindi lo scopo di identificare, analizzare e correggere le criticità in modo strutturato, assicurando una gestione efficace delle anomalie e una continua ottimizzazione dei processi.

3.10.1) Rilevamento

Un'identificazione tempestiva delle problematiche è fondamentale per minimizzare il loro impatto sulle attività[G] . Tutti i membri del team sono incoraggiati a segnalare eventuali anomalie non appena vengono individuate, attraverso strumenti di tracciamento adeguati.
Una volta rilevato un problema, è necessario analizzarne le cause per comprendere la radice della questione ed evitare che si ripresenti in futuro. Per facilitare questo processo[G] , ogni problema deve essere classificato in base alla sua gravità e alla sua urgenza. Inoltre, la documentazione accurata dei problemi e delle loro risoluzioni permette di individuare pattern ricorrenti e definire strategie preventive a lungo termine.

3.10.2) Risoluzione

Dopo aver individuato le cause di un problema, il team procede con l'identificazione della soluzione più efficace. Le strategie di intervento vengono valutate in base all'impatto atteso e all'effort richiesto, cercando sempre di minimizzare eventuali effetti collaterali sulle altre componenti del progetto.
Una volta implementata la soluzione, è necessario verificare che il problema sia stato effettivamente risolto e che non abbia introdotto ulteriori anomalie. Se il problema persiste o emergono nuove criticità, si procede con una revisione della soluzione adottata fino al completo ripristino della funzionalità compromessa.
Il **miglioramento continuo** del processo[G] di risoluzione dei problemi avviene attraverso un'analisi periodica delle segnalazioni raccolte, con l'obiettivo di affinare le pratiche adottate e ridurre il rischio di ricorrenza delle stesse problematiche.

4) Management

4.1) Gestione dell'assegnazione dei ruoli

Il team distribuisce, in accordo con i membri, i ruoli ad ogni periodo$_G$. L'obiettivo è garantire a ciascun componente del gruppo, secondo un criterio di rotazione, l'assegnazione di ogni compito durante lo svolgimento del progetto.
I criteri che vengono considerati ad ogni scelta sono i seguenti:

- disponibilità dei singoli nel periodo$_G$ seguente
- ruoli precedentemente coperti
- tendenza ad alternare i ruoli tra due periodi contigui
- possibilità di lasciare ruoli non coperti se non necessari per la fase successivamente
- possibilità di assegnare uno stesso ruolo a più membri se necessario

Vengono di seguito descritti i 6 ruoli previsti per lo sviluppo del progetto.

4.1.1) Responsabile

La figura di riferimento del gruppo e che lo rappresenta all' esterno$_G$, si occupa del coordinamento e gestione delle risorse.
Nel dettaglio la figura del Responsabile$_G$ si occupa di:

- Organizzare il periodo$_G$ di riferimento, assegnando ruoli e creando issue$_G$
- Monitorare l'andamento del periodo$_G$ in corso mediante analisi della Project board$_G$ e raccogliendo feedback dai diretti interessati
- Organizzare e condurre le riunioni interne del team
- Illustrare, durante i SAL$_G$ periodici con il proponente, il lavoro svolto dal gruppo
- Predisporre il diario di bordo$_G$
- Valutare e gestire i rischi
- Approvare modifiche alla documentazione, secondo l'apposito procedimento
- Stesura del PdP$_G$ con previsioni e retrospettive

4.1.2) Amministratore

Figura con il compito di assicurare l'efficienza, gestione e controllo dell'ambiente IT di lavoro nonché di supporto alla figura del Responsabile$_G$.
Nel dettaglio la figura dell' Amministratore$_G$ si occupa di:

- Controllare e garantire il corretto funzionamento della repository$_G$
- Studiare i processi interni$_G$ per renderli più efficienti
- Garantire la sicurezza della repository$_G$
- Aggiornare il foglio ore relativamente al periodo$_G$ in corso
- Scrittura e aggiornamento delle Norme di Progetto
- Sostituire il Responsabile$_G$ in caso di sua temporanea assenza
- Aggiornare il glossario
- Approvare, dopo la verifica, i documenti redatti o modificati dal Responsabile$_G$

4.1.3) Analista

Figura con il compito di analisi ed illustrazione tecnica del problema. È richiesto, da parte di tale ruolo, la perfetta conoscenza del dominio.
Nel dettaglio la figura dell'Analista si occupa di:

- Studiare il dominio e individuare gli UC$_G$
- Redigere l' AdR$_G$ in tutte le sue sezioni
- Supportare le figure del Progettista e del Programmatore

4.1.4) Progettista

Figura con il compito di individuare e determinare le scelte realizzative. È richiesto, da parte di questa figura, competenze tecniche e tecnologiche aggiornate.

4.1.5) Programmatore

Figura con il compito di seguire la fase di codifica. Ha la responsabilità della realizzazione e mantenimento del codice. Questa figura richiede competenze tecniche ma deleghe limitate.

4.1.6) Verificatore

Figura a supporto di ogni attività$_G$ del progetto. Sono richieste conoscenze e competenze tecniche e la conoscenza dettagliata delle Norme di Progetto del gruppo.
Nel dettaglio la figura del Verificatore$_G$ si occupa di:

- Controllare che la documentazione redatta sia corretta, senza errori ortografici, di contenuto e che rispetti le Norme di Progetto
- Mandare in approvazione i documenti al responsabile$_G$ di progetto

Per le attività$_G$ in capo a tale figura si rimanda al procedimento per la gestione delle modifiche della documentazione - Sezione 3.5.1

4.2) Gestione della board

Il team utilizza la board di GitHub$_G$ per la gestione delle issue$_G$ e delle attività$_G$.
Essa è suddivisa in colonne, ognuna delle quali rappresenta uno stato dell' attività$_G$.

1. **To Do**: rappresenta il nostro backlog$_G$, ovvero tutte le attività$_G$ che devono essere svolte
2. **In Progress**: attività$_G$ in corso di svolgimento
3. **In review**: attività$_G$ completata e in attesa di verifica
4. **Done**: attività$_G$ completata e verificata

4.2.1) Processo di utilizzo board

1. Assegnazione di un' attività$_G$: l' attività$_G$ viene assegnata a un membro del team
    - Se c'è la presenza di un «sottogruppo» di lavoro, le decisioni relative al come suddividere le attività$_G$ saranno a carico del « responsabile$_G$ « del sottogruppo. Si attua quindi una sorta di «divide et impera» per garantire una maggiore efficienza e una migliore gestione delle attività$_G$.
2. Inizio dell' attività$_G$: il membro del team assegnato sposta l' attività$_G$ dalla colonna **To Do** a **In Progress**
3. Completamento dell' attività$_G$: il membro del team sposta la card dalla colonna **In Progress** a **In review**
4. Verifica$_G$ dell' attività$_G$: il verificatore$_G$ controlla la Pull Request$_G$ associata all' attività$_G$ e, se viene approvata, per la struttura data alla repository$_G$,l' attività$_G$ verrà spostata in automatico da **In review** a **Done**

Sarà compito del responsabile$_G$ del progetto$_G$ controllare che le attività$_G$ siano assegnate correttamente e che la board sia aggiornata. Inoltre, assegnerà il grado di priorità, in modo da garantire che quelle più importanti siano svolte per prime.

4.3) Gestione e Analisi delle ore di lavoro

La gestione delle ore di lavoro e dei relativi costi è uno degli aspetti fondamentali per monitorare l'andamento del progetto. A tal fine, ogni membro del team dispone di una sezione del foglio ore creato tramite Google Sheets$_G$, che permette di registrare, riepilogare e analizzare le ore svolte e i costi associati. Inoltre, i dati inseriti nel foglio sono integrati con Grafana$_G$, un servizio che fornisce un cruscotto$_G$ di monitoraggio visivo e analitico. Di seguito sono spiegate in dettaglio le diverse sezioni e funzionalità$_G$.

4.3.1) Struttura e utilizzo del foglio ore

Il foglio ore si compone di due parti principali:

1. **Riepilogo dei Costi** (sezione sinistra)
   - Questa sezione fornisce un quadro complessivo delle ore totali e dei costi associati ai vari ruoli svolti nei vari periodi.
   - **Colonne principali**:
     - **Ruolo**: elenca i ruoli ricoperti (es. Responsabile$_G$, Amministratore$_G$, Verificatore$_G$, ecc.).
     - **Periodo$_G$ X**: numero di ore svolte per ruolo nel periodo$_G$ X.
     - **Tot. h**: somma delle ore svolte per ciascun ruolo.
     - **€/ora**: tariffa oraria di ciascun ruolo.
     - **Costo**: calcolo del costo totale ottenuto moltiplicando le ore svolte per la tariffa oraria.
   - Questa sezione consente di monitorare immediatamente i costi associati a ciascun ruolo e verificare se i tempi e i budget sono in linea con le previsioni.
2. **Tabella Oraria per Periodo$_G$** (sezione destra)
   - La sezione a destra è suddivisa in **periodi di riferimento**, ciascuno indicato con un numero progressivo e date specifiche.
   - **Colonne principali**:
     - **Data**: rappresenta il giorno specifico per cui vengono registrate le ore.
     - **Ruoli**: ciascun ruolo ha una colonna dedicata (es. Responsabile$_G$, Amministratore$_G$, Verificatore$_G$, ecc.).
     - **Ore svolte**: l'utente inserisce manualmente le ore svolte per ogni ruolo, in corrispondenza della data indicata.
   - Al termine del periodo$_G$, il totale delle ore inserite viene calcolato automaticamente e riportato nel riepilogo a sinistra, fornendo una chiara visione delle ore effettivamente lavorate.

4.3.2) Integrazione con Grafana

I dati raccolti nel foglio ore sono automaticamente collegati a Grafana$_G$, un servizio di monitoraggio che permette di visualizzare l'andamento del progetto attraverso grafici e dashboard interattive. Questo collegamento permette di avere un'analisi più approfondita e visiva delle attività$_G$ svolte. La dashboard di Grafana$_G$ è suddivisa in diverse sezioni chiave:

1. **Stato delle Issue$_G$**
   - Grafana$_G$ monitora le attività$_G$ del repository$_G$ GitHub$_G$, mostrando:

        - **Issue**$_G$ **aperte e chiuse** in un grafico a torta.
        - **Issue**$_G$ **in corso**, **in revisione**$_G$ **e completate** evidenziando il loro stato attuale.
2. Riepilogo di tutte le **pull request**$_G$ e in che stato si trovano.
3. Stato della **board** di GitHub$_G$ (**4.2 Gestione della board**).
4. **Andamento Ore e Costi**
    - Questa sezione fornisce un confronto visivo tra le **ore previste** e le **ore effettivamente** svolte per ciascun ruolo.
    - Il grafico in basso a sinistra evidenzia:
        - **Ore Previste**: rappresentate in giallo.
        - **Ore Effettive**: rappresentate in verde.
5. **Andamento Costi Preventivati vs Effettivi**
    - In basso a destra, un grafico a linea mostra:
        - **Costi Preventivati**: i costi stimati durante la pianificazione del periodo$_G$.
        - **Costi Effettivi**: i costi registrati a fine periodo$_G$.
    - Questo confronto permette di valutare eventuali scostamenti rispetto al piano iniziale e di adottare misure correttive.
6. **Riepilogo delle Attività**$_G$
    - Nella sezione centrale una tabella riporta i ruoli assegnati per il periodo$_G$ corrente.

4.4) Norme tipografiche

I documenti devono rispettare standard tipografici e sintattici uniformi per garantire chiarezza e coerenza. Di seguito, si riportano le regole principali da seguire.

4.4.1) Regole Sintattiche

4.4.1.1) Nomi dei file

- I documenti iniziano con una lettera maiuscola.
- Il nome del documento è composto dalle parole che indicano il tipo e l'argomento principale del documento. Se il nome è formato da più parole, queste devono essere separate da spazi (es. Norme di Progetto, Piano di Progetto).
- I verbali seguono il formato AAAA-MM-GG, dove AAAA-MM-GG rappresenta la data dell'incontro a cui il verbale si riferisce.

4.4.1.2) Stili del testo

- **Grassetto**: evidenzia informazioni chiave come definizioni, titoli di sezioni o termini importanti.
- **Corsivo**: evidenzia parole tecniche o concetti introdotti per la prima volta.
- **Glossario**: i termini inseriti nel glossario sono contrassegnati da una G blu in pedice. Ad esempio, il termine verificatore$_G$ appare con una G blu sotto di esso.
- **Link**: i collegamenti ipertestuali sono visualizzati in blu, come nel caso del link nella sezione 1.3
- **Titoli**: seguono una gerarchia fino al livello H4, con formattazione coerente (H1, H2, H3, H4).
- **Font e dimensioni**: il font scelto è Roboto Serif, con una dimensione di 12 pt per il corpo del testo, e interlinea 1,5.
- **Margini**: i margini sono impostati a 2 cm sui lati orizzontali e 2,5 cm sui lati verticali.
- **Elenchi**:

- **Elenchi puntati**: devono essere usati per elencare oggetti, idee o concetti che non seguono un ordine particolare. Ad esempio, per elencare requisiti, caratteristiche, o attività[G] che non sono sequenziali.
- **Elenchi numerati**: devono essere utilizzati quando si descrivono attività[G] che devono essere eseguite in un ordine preciso, come per le procedure passo passo, le istruzioni sequenziali o le fasi di un processo.

4.5) Gestione immagini

- Ogni immagine presente nei documenti deve essere salvata nella cartella *img*
- Per ogni documento che utilizzi immagini non condivise con altri documenti, è necessario creare una sottocartella con il nome del suddetto documento in cui inserire le immagini ad esso relative

5) Processi organizzativi

È fondamentale che il gruppo sia allineato nelle tempistiche e modalità di organizzazione dei processi nell'ottica di una corretta gestione dei task$_G$ ed eventuali rischi annessi.

5.1) Gestione dei processi

Un processo$_G$ è un insieme di attività$_G$ correlate e coese che trasformano bisogni (input) in prodotti (output) secondo specifiche regole.

L'intero ciclo di vita$_G$ di ogni processo è supportato dalla gestione di questo mediante il sistema di Issue$_G$ di Github$_G$.

La gestione di un processo è composta da diverse fasi:

1. Identificazione e definizione
2. Pianificazione
3. Monitoraggio
4. Gestione dei rischi
5. Retrospettiva$_G$

5.1.1) Identificazione e definizione di processi

Elemento fondamentale per la gestione di un processo è l'identificazione di questo. Un processo viene indicato come una minima attività$_G$ che compone il progetto, indipendentemente essa sia di progettazione, analisi, codifica o gestione/amministrazione del progetto stesso.

5.1.1.1) Identificazione mediante sistema Issue di Github

Ogni processo viene identificato da:

- **ID**$_G$, generato automaticamente dal sistema
- **Nome**
- **Descrizione**, se necessaria
- **Membro** (o membri) del team assegnati
- **Label**$_G$, fondamentale per identificare l'appartenenza del processo. Ogni label$_G$ si riferisce alla relativa parte di documentazione/codifica di cui il processo fa parte. Nel dettaglio:
  - AdR$_G$
  - Agg_sito
  - Candidatura
  - Fix, per indicare la correzione di un errore ed è obbligatorio associare una seconda label$_G$ che identifichi l'appartenenza del processo
  - Glossario
  - NdP$_G$
  - PdP$_G$
  - PdQ$_G$
  - V.E.
  - V.I.
- **Progetto**, configurazione di Github$_G$ necessaria per poter gestire la issue$_G$ mediante la Project Board$_G$
- **Milestone**$_G$, per identificare il periodo$_G$ a cui il processo è associato

5.1.2) Pianificazione

5.1.2.1) Descrizione

Ogni processo viene associato ad un periodo$_G$, indicato nel sistema di Issue$_G$ come Milestone$_G$. Tale associazione consente di identificare il processo dentro una fase, definita da una data di inizio ed una fine, definendo quindi un termine massimo di completamento, salvo specifica indicazione a preventivo o motivazione a consuntivo. Tale gestione consente inoltre di avere una visione su tutti i processi, consentendo il monitoraggio e la retrospettiva$_G$ del periodo$_G$ stesso con stime di tempi, risorse e costi necessari per il completamento delle Issue$_G$.

5.1.2.2) Attività a cura del responsabile

Il responsabile di progetto è tenuto a:

1. creare la issue$_G$ in Github, assegnando un titolo esaustivo
2. inserire, se necessario, una descrizione procedurale volta a dare maggiori dettagli
3. assegnare la issue al membro/ai membri del team responsabili del completamento della stessa
4. assegnare una o più labels$_G$ volte a categorizzare le issue$_G$
5. assegnare la issue al progetto
6. assegnare la issue alla Milestone$_G$

5.1.2.3) Diagramma di Gantt

Per una corretta pianificazione e gestione delle issue$_G$, è compito di ogni assegnatario delle issue$_G$:

1. impostare la *start date* e la *end date* nella relative sezione di Github$_G$, in modo tale che la project board mostri automaticamente il diagramma di Gantt con le tempistiche di inizio e fine del processo.

5.1.3) Monitoraggio

5.1.3.1) Descrizione

È necessario conoscere, in ogni momento, lo stato di avanzamento del processo mediante un corretto utilizzo della Project Board$_G$ di Github$_G$.
La board permette al responsabile$_G$ di progetto di intervenire tempestivamente in caso di problematiche che sono sorte o stanno per sorgere.
È compito del responsabile$_G$ di progetto interfacciarsi con l'assegnatario della Issue$_G$ qualora si presentasse qualche situazione di rischio per trovare una soluzione a questa.
Se un membro del gruppo nota difficoltà non previste durante lo svolgimento del processo è tenuto ad avvisare tutto il team e sarà cura del responsabile$_G$ trovare una soluzione al problema presentato.

5.1.3.2) Attività a cura dell'assegnatario di una issue

Ogni Issue$_G$ appartiene ad uno stato, in tempo reale, che rappresenta il processo. È a cura dell'assegnatario della Issue$_G$ identificare e aggiornare lo stato del processo mediante trascinamento nella Project Board$_G$ nello stato corretto:

- **Todo**$_G$, Issue$_G$ creata ma non ancora iniziata
- **In progress**, Issue$_G$ in lavorazione
- **In review**, Issue$_G$ completata e in attesa di verifica
- **Done**, Issue$_G$ terminata

5.1.4) Gestione dei rischi

Ogni processo può essere soggetto a rischi, indicati nel Piano di Progetto. Una corretta prevenzione e gestione dei rischi, come indicato al punto precedente, richiede il corretto e tempestivo aggiornamento di una board.
Il responsabile$_G$ di progetto, al verificarsi di una situazione di rischio, è tenuto a prendere decisioni volte all'eliminazione di tale rischio con l'obiettivo di terminare i processi nei tempi previsti e rispettando le procedure ed indici di qualità. Tali decisioni vengono indicate e motivate nei verbali interni$_G$ e nel PdP$_G$, in quest'ultimo nella sezione di retrospettiva$_G$ del periodo$_G$.

5.1.5) Retrospettiva

Ogni singolo processo è parte integrante della retrospettiva$_G$ del periodo$_G$, dove eventuali criticità devono essere evidenziate e giustificate.
Durante l'incontro periodico SAL$_G$ con il proponente viene relazionata, da coloro che hanno seguito i processi interessati dalla riunione, la retrospettiva$_G$ del processo stesso.

5.1.6) Miglioramento

L' attività$_G$ di miglioramento è un aspetto fondamentale per garantire efficienza e qualità dei processi. Si identificano tutte quelle «aree» che necessitano degli accorgimenti che possono interessare questioni tempistiche e/o di qualità apportando quindi le dovute modifiche necessarie. L'identificazione si materializza attraverso la retrospettiva$_G$ svolta durante ogni riunione (ampiamente descritto all' interno$_G$ del PdP$_G$) e anche grazie al cruscotto$_G$ che permette di avere una visione d'insieme dei processi e delle issue$_G$ in corso.

5.1.7) Formazione

L' attività$_G$ di formazione è necessaria per avere un gruppo con conoscenze allineate e adatte per la realizzazione del progetto. L'obiettivo dell' attività$_G$ è un gruppo con membri capaci di ricoprire ogni ruolo all' interno$_G$ del gruppo avendo conseguito un livello di conoscenza generale adatto.
L' attività$_G$ si divide in due modalità:

- **Individuale**: ogni membro si impegnerà a raggiungere un livello di conoscenze adatto per ricoprire ogni ruolo in modo autonomo
- **Di gruppo**: per gruppo si intende anche due persone, nel caso l' attività$_G$ individuale non sia consigliata o normalmente essa si svolge nel cambio dei ruoli passando le conoscenze maturate durante il periodo$_G$ al gruppo o al singolo che andrà a ricoprire il ruolo.

6) Standard di qualità

Per una corretta gestione del ciclo di vita$_G$ del progetto e per garantire la qualità dei processi e del prodotto software verranno adottati i seguenti standard internazionali sviluppati dall' ISO$_G$:

- **ISO/IEC 9126**$_G$: lo standard per la valutazione della qualità del prodotto software. Questo modello consente di analizzare e valutare il software in base a caratteristiche fondamentali, quali funzionalità$_G$, affidabilità, usabilità, efficienza, manutenibilità e portabilità. Queste caratteristiche sono misurabili attraverso delle metriche. La scelta di questo standard riflette l'obiettivo di fornire un prodotto che soddisfi pienamente le specifiche richieste del progetto.
- **ISO/IEC 12207:1995**$_G$: lo standard per il ciclo di vita$_G$ del software, che definisce un insieme strutturato di processi per la gestione e lo sviluppo del progetto. Questo standard prevede la suddivisione in processi primari, di supporto e organizzativi, garantendo una visione completa e coerente della gestione delle attività$_G$ durante l'intero ciclo di vita$_G$ del progetto.

Questa combinazione di standard consente di bilanciare l'attenzione sulla qualità del prodotto con un approccio metodico alla gestione dei processi, assicurando un risultato finale che sia funzionale, efficiente e conforme alle migliori pratiche internazionali.

6.1) Modello di qualità secondo Standard ISO/IEC 9126

Di seguito vengono elencate e descritte le categorie fondamentali che classificano il modello. Ciascuna di queste è caratterizzata a sua volta da delle sotto-caratteristiche. Una delle sotto-caratteristiche comuni a tutte le categorie è la **conformità**, che si riferisce alla capacità del software di rispettare standard tecnici, norme e regolamenti relativi alla specifica caratteristica.

6.1.1) Funzionalità

É la capacità del software di fornire funzioni che soddisfino i requisiti specificati o impliciti, con sotto-caratteristiche come:

- **Adeguatezza**: è la capacità del software di fornire un appropriato insieme di funzioni per soddisfare i requisiti specifici dell'utente.
- **Accuratezza**: è la capacità del software di fornire i risultati corretti in modo tale che corrispondano a quanto richiesto.
- **Interoperabilità**: è la capacità del software di interagire ed operare con altri sistemi.
- **Sicurezza**: è la capacità del software di proteggere informazioni e dati da accessi non autorizzati.

6.1.2) Affidabilità

Misura la capacità del software di mantenere un livello di prestazioni specificato in determinate condizioni, anche in presenza di errori, per un periodo$_G$ di tempo stabilito. Le sue sotto-caratteristiche sono:

- **Maturità**: è la capacità del software di gestire in modo stabile le operazioni, evitando errori, malfunzionamenti e risultati non corretti.
- **Tolleranza agli errori**: è la capacità di mantenere prestazioni specificate anche in caso di errori.
- **Recuperabilità**: è la capacità del software di ripristinare un livello appropriato di prestazioni in caso di errori.

6.1.3) Usabilità

É la capacità del software di essere compreso, appreso e utilizzato dall'utente in condizioni specifiche. Le sotto-caratteristiche dell'usabilità sono:

- **Comprensibilità**: è la facilità con la quale l'utente può capire le funzionalità[G] disponibili e come queste possono essere utilizzate per raggiungere i propri obiettivi. Include la chiarezza dell'interfaccia utente e delle informazioni presentate.
- **Apprendibilità**: è la facilità con la quale gli utenti possono apprendere come utilizzare il sistema.
- **Operabilità**: è la capacità del software di consentire agli utenti di operare e controllare il sistema senza difficoltà.
- **Attrattiva**: è la capacità del software di essere gradevole per l'utente che ne fa uso, attraverso elementi di piacevolezza come aspetti grafici o interattivi.

6.1.4) Efficienza

É la capacità del software di fornire prestazioni adeguate rispetto alle risorse utilizzate, in condizioni specifiche. Le sotto-caratteristiche dell'efficienza sono:

- **Comportamento temporale**: è la capacità di fornire adeguati tempi di risposta ed elaborazione durante l'esecuzione.
- **Utilizzo delle risorse**: è la capacità del software di utilizzare la quantità appropriata di risorse del sistema.

6.1.5) Manutenibilità

É la facilità con cui un sistema software può essere modificato per correggere difetti e migliorare le prestazioni. Le sue sotto-caratteristiche sono:

- **Analizzabilità**: è la facilità con la quale è possibile analizzare il codice per localizzare un errore o un difetto nello stesso.
- **Modificabilità**: è la facilità con cui il software può essere modificato per aggiungere nuove funzionalità[G] o per cambiare quelle esistenti.
- **Stabilità**: è la capacità del software di evitare nuovi errori o difetti durante o dopo una modifica.
- **Testabilità**: è la facilità con cui il software può essere testato per verificare la correttezza delle modifiche. É fondamentale per garantire che non vegano introdotti difetti e che il sistema funzioni come previsto.

6.1.6) Portabilità

É la facilità con cui un software può essere trasferito da un ambiente a un altro. Le sue sotto-caratteristiche sono:

- **Adattabilità**: è la capacità del software di essere modificato per adattarsi a nuovi ambienti senza necessitare di grandi modifiche.
- **Installabilità**: è la facilità con cui il software può essere installato in un nuovo ambiente.
- **Coesistenza**: è la capacità del software di funzionare correttamente insieme ad altri sistemi o software già presenti nello stesso ambiente, senza causare conflitti o interferenze.
- **Sostituibilità**: é la facilità con cui il software può sostituire o essere sostituito da altre applicazioni nello stesso ambiente.

6.2) Suddivisione dei processi secondo Standard ISO/IEC 12207:1995

6.2.1) Processi primari

Sono i processi che comprendono le attività[G] direttamente legate allo sviluppo del software, si occupano quindi della realizzazione, distribuzione e manutenzione del prodotto software.
L'obiettivo di questi processi è garantire che il prodotto sia consegnato e mantenuto secondo i requisiti.

6.2.2) Processi di supporto

Sono i processi che includono la gestione dei documenti e dei processi di controllo della qualità, dunque non producono direttamente il software, ma forniscono attività[G] e servizi necessari per garantire la qualità ed efficacia.
L'obiettivo di questi processi è garantire che i processi primari funzionino in modo fluido e il prodotto finale soddisfi gli standard richiesti.

6.2.3) Processi organizzativi

Sono i processi che coprono gli aspetti manageriali e di gestione delle risorse dunque forniscono la struttura e le pratiche a livello organizzativo per gestire e migliorare i processi primari e di supporto.
L'obiettivo di questi processi è garantire che l'organizzazione sia in grado di supportare lo sviluppo, la gestione e il miglioramento continuo dei processi e dei prodotti.

7) Metriche di qualità

Le metriche di qualità sono misure oggettive e quantificabili per valutare e monitorare la qualità dei processi di sviluppo e del prodotto software. Servono dunque a garantire che il software soddisfi i requisiti richiesti e rispetti gli standard di qualità stabiliti, inoltre permettono di controllare l'andamento del progetto e quindi di individuare eventuali aree critiche dove adottare procedure di miglioramento.
Le due categorie principali sono:

- **metriche di qualità del processo**: valutano la qualità dei processi di sviluppo e gestione;
- **metriche di qualità del prodotto**: valutano direttamente le caratteristiche del software.

7.1) Metriche di qualità del processo

Ogni metrica di questa sezione verrà identificata come segue:

MPC.NumProgressivo

Dove **MPC** è l'abbreviazione di **M**etriche di qualità del **P**ro**C**esso e **NumProgressivo** è un intero che aumenta con ogni metrica.
Le formule di alcune di queste metriche utilizzano il valore BAC_G , che sta per *Budget At Completion*, ossia il costo totale pianificato per completare il progetto.

7.1.1) Processi primari

- **MPC1**:
  - **Nome**: Schedule Adherence (**SA**)
  - **Descrizione**: percentuale di attività$_G$ completate entro le scadenze stabilite.
  - **Obiettivo**: misurare quanto il progetto rispetta i tempi previsti.
  - **Formula**: $SA = \frac{\text{attività completate in tempo}}{\text{attività pianificate}}\ x\ 100$
- **MPC2**:
  - **Nome**: Earned Value (**EV**)
  - **Descrizione**: rappresenta il valore del lavoro effettivamente completato fino a quel periodo$_G$.
  - **Obiettivo**: misurare il progresso del progetto.
  - **Formula**: $EV = lavoro\ completato(\%)\ x\ BAC_G$
- **MPC3**:
  - **Nome**: Planned Value (**PV**)
  - **Descrizione**: rappresenta il valore del lavoro pianificato da completare entro una determinata data.
  - **Obiettivo**: determinare quanto lavoro dovrebbe essere completato in un certo momento del progetto.
  - **Formula**: $PV = lavoro\ pianificato(\%)\ x\ BAC_G$
- **MPC4**:
  - **Nome**: Schedule Variance (**SV**)
  - **Descrizione**: differenza tra il valore del lavoro completato e quello pianificato fino a un dato momento del progetto.
  - **Obiettivo**: rilevare la presenza di ritardi rispetto al piano iniziale, permettendo di intervenire tempestivamente.

- **Formula**: *SV = EV - PV*

- **MPC5**:
  - **Nome**: Actual Cost (**AC**)
  - **Descrizione**: rappresenta il costo effettivo sostenuto per il lavoro completato fino a una data specifica.
  - **Obiettivo**: fornire una visione chiara delle spese effettive.
  - **Formula**: *somma dei costi effettivi* (dato ricavabile dall'esito dei periodi nel PdP_G)
- **MPC6**:
  - **Nome**: Cost Performance Index (**CPI**)
  - **Descrizione**: misura l'efficienza dei costi di un progetto, confrontando il valore guadagnato (EV) con il costo effettivo (AC).
  - **Obiettivo**: fornire un'indicazione dell'efficacia nell'uso delle risorse economiche.
  - **Formula**: $CPI = \frac{\text{EV}}{\text{AC}}$
- **MPC7**:
  - **Nome**: Cost Variance (**CV**)
  - **Descrizione**: misura la differenza tra il valore guadagnato (EV) e il costo effettivo (AC), un valore negativo indica che si è oltre il budget previsto, mentre un valore positivo indica un risparmio rispetto ai costi pianificati.
  - **Obiettivo**: monitorare l'aderenza al budget pianificato.
  - **Formula**: *CV = EV - AC*
- **MPC8**:
  - **Nome**: Estimated At Completion (**EAC**)
  - **Descrizione**: stima il costo totale previsto per completare il progetto, tenendo conto del rendimento attuale.
  - **Obiettivo**: fornire una previsione aggiornata dei costi finali del progetto.
  - **Formula**: $EAC = \frac{\text{BAC}}{\text{CPI}}$
- **MPC9**:
  - **Nome**: Estimated To Complete (**ETC**)
  - **Descrizione**: rappresenta la stima dei costi necessari per completare il lavoro rimanente di un progetto.
  - **Obiettivo**: fornire una previsione dei costi futuri.
  - **Formula**: *ETC = EAC - AC*

7.1.2) Processi di supporto

- **MPC10**:
  - **Nome**: Percentuale di Casi di Test Superati (**PCTS**)
  - **Descrizione**: misura la percentuale di casi di test eseguiti che hanno avuto esito positivo rispetto al totale dei casi di test eseguiti.
  - **Obiettivo**: valutare l'efficacia dei test nel garantire che il software soddisfi i requisiti e non presenti errori.
  - **Formula**: $PCTS = \frac{\text{casi di test superati}}{\text{casi di test eseguiti}}$ *x 100*
- **MPC11**:

- **Nome**: Percentuale di Metriche Soddisfatte (**PMS**)
- **Descrizione**: misura la percentuale di metriche di qualità che sono state soddisfatte.
- **Obiettivo**: valutare in modo globale il livello di qualità raggiunto.
- **Formula**: $PMS = \frac{\text{metriche soddisfatte}}{\text{metriche totali}}\ x\ 100$

7.1.3) Processi organizzativi

- **MPC12**:
  - **Nome**: Rischi Non Previsti (**RNP**)
  - **Descrizione**: rappresenta il numero di rischi non previsti rilevati durante il progetto.
  - **Obiettivo**: ridurre il numero di rischi imprevisti, migliorando pianificazione e gestione.

7.2) Metriche di qualità del prodotto

Ogni metrica di questa sezione verrà identificata come segue:

MPD.NumProgressivo

Dove **MPD** è l'abbreviazione di **M**etriche di qualità del **Pro**d**o**tto e **NumProgressivo** è un intero che aumenta con ogni metrica.

7.2.1) Funzionalità

- **MPD1**:
  - **Nome**: Requisiti Obbligatori Soddisfatti (**ROBS**)
  - **Descrizione**: percentuale di requisiti obbligatori soddisfatti.
  - **Obiettivo**: valutare il grado di soddisfacimento dei requisiti richiesti per il progetto.
  - **Formula**: $ROBS = \frac{\text{requisiti obbligatori soddisfatti}}{\text{requisiti obbligatori totali}}\ x\ 100$
- **MPD2**:
  - **Nome**: Requisiti Desiderabili Soddisfatti (**RDS**)
  - **Descrizione**: percentuale di requisiti desiderabili soddisfatti.
  - **Obiettivo**: valutare il grado di soddisfacimento dei requisiti richiesti per il progetto.
  - **Formula**: $ROS = \frac{\text{requisiti desiderabili soddisfatti}}{\text{requisiti desiderabili totali}}\ x\ 100$
- **MPD3**:
  - **Nome**: Requisiti Opzionali Soddisfatti (**ROPS**)
  - **Descrizione**: percentuale di requisiti opzionali soddisfatti.
  - **Obiettivo**: valutare il grado di soddisfacimento dei requisiti richiesti per il progetto.
  - **Formula**: $ROS = \frac{\text{requisiti opzionali soddisfatti}}{\text{requisiti opzionali totali}}\ x\ 100$

7.2.2) Affidabilità

- **MPD4**:
  - **Nome**: Code Coverage (**CC**)
  - **Descrizione**: percentuale di codice coperto dai test rispetto al totale del codice sviluppato.
  - **Obiettivo**: valutare l'efficacia del processo di testing e il livello di qualità assicurato dal processo.
  - **Formula**: $CC = \frac{\text{linee di codice testate}}{\text{linee di codice totali}}\ x\ 100$
- **MPD5**:
  - **Nome**: Indice Gulpease (**MIG**)

    - **Descrizione**: è una metrica che misura la leggibilità di un testo in lingua italiana, basandosi sulla lunghezza delle parole e delle frasi.
    - **Obiettivo**: garantire che la documentazione prodotta sia chiara e accessibile$_G$, evitando testi complessi.
    - **Formula**: $MIG = 89 + \frac{300 \cdot (\text{numero frasi}) - 10 \cdot (\text{numero lettere})}{\text{numero parole}}$

- **MPD6**:
    - **Nome**: Failure Density (**FD**)
    - **Descrizione**: rappresenta il numero di errori rilevati per unità$_G$ di codice.
    - **Obiettivo**: valutare la qualità del codice e ridurre il numero di errori.
    - **Formula**: $FD = \frac{\text{numero errori rilevati}}{\text{linee di codice totali}} \ x\ 100$

- **MPD7**:
    - **Nome**: Statement Coverage (**SC**)
    - **Descrizione**: indica la percentuale di istruzioni del codice eseguite almeno una volta durante i test.
    - **Obiettivo**: misurare la copertura dei test.
    - **Formula**: $SC = \frac{\text{istruzioni eseguite}}{\text{istruzioni totali}} \ x\ 100$

- **MPD8**:
    - **Nome**: Branch$_G$ Coverage (**BC**)
    - **Descrizione**: misura la percentuale di ramificazioni eseguite almeno una volta durante i test rispetto al totale delle ramificazioni presenti nel codice.
    - **Obiettivo**: garantire che tutte le possibili diramazioni del codice siano state testate per identificare eventuali errori nei percorsi condizionali.
    - **Formula**: $BC = \frac{\text{ramificazioni eseguite}}{\text{ramificazioni totali}} \ x\ 100$

- **MPD9**:
    - **Nome**: Correttezza Ortografica (**CO**)
    - **Descrizione**: misura il numero di errori ortografici presenti all' interno$_G$ di un documento del progetto.
    - **Obiettivo**: garantire una documentazione chiara, migliorando la leggibilità.

7.2.3) Usabilità

- **MPD10**:
    - **Nome**: Facilità di Utilizzo (**FU**)
    - **Descrizione**: valuta quanto il software sia semplice da utilizzare per gli utenti.
    - **Obiettivo**: garantire un'esperienza utente positiva.

- **MPD11**:
    - **Nome**: Tempo di Apprendimento (**TA**)
    - **Descrizione**: misura il tempo necessario affinché un nuovo utente apprenda come utilizzare il sistema.
    - **Obiettivo**: ridurre il tempo richiesto per apprendere le funzionalità$_G$ del sistema.

7.2.4) Efficienza

- **MPD12**:
    - **Nome**: Tempo Medio di Risposta (**TMR**)

- **Descrizione**: misura il tempo medio che il sistema impiega per rispondere a una richiesta .
- **Obiettivo**: ottimizzare i tempi di risposta per garantire una migliore esperienza utente.
- **Formula**: $TMR = \frac{\text{somma tempi di risposta delle richieste}}{\text{numero totale di richieste}}$

- **MPD13**:
  - **Nome**: Utilizzo delle Risorse (**UR**)
  - **Descrizione**: indica l'efficienza del software in termini di utilizzo delle risorse durante l'esecuzione.
  - **Obiettivo**: monitorare l'efficienza nell'uso delle risorse.

7.2.5) Manutenibilità

- **MPD14**:
  - **Nome**: Complessità Ciclomatica (**V(G)**)
  - **Descrizione**: misura la complessità di un programma in base al numero di percorsi indipendenti nel flusso di controllo del codice.
  - **Obiettivo**: fornire un'indicazione della difficoltà di test e manutenzione del codice, identificando eventuali porzioni di codice troppo complesse.
  - **Formula**: *V(G) = E - N + 2P*
    dove:
    - E: numero di archi del grafo (transizioni tra nodi);
    - N: numero di nodi del grafo (istruzioni o blocchi di codice);
    - P: numero di componenti connesse (numero di funzioni o moduli del programma).