

Norme di progetto

versione 2.0.0



7commits@gmail.com

Progetto di Ingegneria del Software

A.A. 2024/2025

Destinatari	Responsabile	Redattori	Verificatori
Prof. Tullio Vardanega	Marco Cola	Stefano Dal Poz	Giulia Hu
Prof. Riccardo Cardin		Marco Cola	Ruize Lin
Gruppo 🇮🇹 7Commits		Giada Rossi	Marco Cola
			Mattia Piva
			Giada Rossi

Registro delle modifiche

Versione	Data	Autori	Verificatori	Descrizione
v2.0.0	2025-08-24	Marco Cola	Stefano Dal Poz, Ruize Lin	Approvazione per PB
v1.1.1	2025-08-23	Marco Cola, Stefano Dal Poz	Ruize Lin	Migliorata sezione «Testing»
v1.1.0	2025-08-20	Marco Cola	Stefano Dal Poz, Ruize Lin	Corrette spaziature tra sezioni ed elenchi puntati, ridotta la profondità della numerazione delle sezioni (massimo 4 livelli)
v1.0.2	2025-08-18	Stefano Dal Poz	Marco Cola	Aggiunta sezione «Testing»
v1.0.1	2025-08-14	Stefano Dal Poz	Marco Cola	Aggiunta sezione «Codifica»
v1.0.0	2025-06-12	Marco Cola	Ruize Lin	Approvazione per RTB
v0.1.5	2025-05-16	Stefano Dal Poz	Ruize Lin	Aggiunti collegamenti al Glossario
v0.1.4	2025-05-10	Giulia Hu	Ruize Lin	Ampliato sezione 2.2.1
v0.1.3	2025-05-06	Stefano Dal Poz	Michele Ogniben, Ruize Lin	Aggiunte sezioni 3.1.6 e 3.1.7
v0.1.2	2025-05-05	Marco Cola	Michele Ogniben, Ruize Lin	Aggiunta sezione 4.1.3
v0.1.0	2025-05-05	Stefano Dal Poz	Michele Ogniben, Marco Cola	Ampliato paragrafo Analisi dei requisiti in sezione 2.2
v0.0.10	2025-05-02	Stefano Dal Poz	Michele Ogniben, Marco Cola, Ruize Lin	Aggiunti paragrafi Progettazione, Codifica e Testing in sezione 2; sezione 2.2 e paragrafo Miglioramento, Formazione in sezione 4
v0.0.9	2025-04-28	Stefano Dal Poz	Michele Ogniben, Marco Cola	Aggiunti paragrafo Verifica e Validazione in sezione 3
v0.0.8	2025-04-24	Stefano Dal Poz	Michele Ogniben	Aggiunte sezioni 5 e 6
v0.0.7	2025-04-19	Giada Rossi	Marco Cola, Michele Ogniben	Aggiunto sezione 5.1 Gestione organizzativa
v0.0.6	2025-04-18	Marco Cola	Marco Cola, Giada Rossi	Aggiunta descrizione ruoli sprint in 3.1.3.6

v0.0.5	2025-04-14	Stefano Dal Poz, Marco Cola	Marco Cola, Giada Rossi	Aggiunti collegamenti al Glossario
v0.0.4	2025-04-10	Giada Rossi	Marco Cola	Aggiunto paragrafo Strumenti + capitolo Verifica
v0.0.3	2025-04-08	Stefano Dal Poz	Marco Cola	Aggiunto capitolo Processi di supporto
v0.0.2	2025-04-01	Stefano Dal Poz	Marco Cola	Aggiunto capitolo Introduzione
v0.0.1	2025-03-31	Stefano Dal Poz	Marco Cola	Stesura iniziale del documento

Indice

1. Introduzione	6
1.1. Scopo del documento	6
1.2. Scopo del progetto	6
1.3. Glossario	6
1.4. Riferimenti	6
1.4.1. Riferimenti normativi	6
1.4.2. Riferimenti informativi	6
2. Processi primari	7
2.1. Fornitura	7
2.1.1. Scopo	7
2.1.2. Documentazione da consegnare	7
2.1.2.1. Analisi dei Requisiti	7
2.1.2.2. Piano di Progetto	7
2.1.2.3. Piano di Qualifica	7
2.1.2.4. Glossario	8
2.2. Sviluppo	8
2.2.1. Analisi dei requisiti	8
2.2.1.1. Implementazione	8
2.2.1.2. Casi d'uso	8
2.2.1.3. Diagramma casi d'uso	9
2.2.1.4. Notazione dei casi d'uso	12
2.2.1.5. Requisiti: Notazione	12
2.2.1.6. Requisiti: Suddivisione	12
2.2.2. Progettazione	13
2.2.2.1. Documentazione	13
2.2.2.2. Fasi della progettazione	13
2.2.2.3. Progettazione logica	13
2.2.2.4. Progettazione di dettaglio	13
2.2.2.5. Diagrammi delle classi	13
2.2.2.6. Strumenti	14
2.2.3. Codifica	14
2.2.4. Testing	14
2.2.4.1. Tipologie di test	14
2.2.4.2. Notazione dei test	15
2.2.4.3. Strumenti	15
3. Processi di supporto	16
3.1. Documentazione	16
3.1.1. Scopo	16
3.1.2. Strumento utilizzato	16
3.1.3. Struttura dei documenti	16
3.1.3.1. Pagina di copertina	16
3.1.3.2. Registro delle modifiche	16
3.1.3.3. Indice	17
3.1.3.4. Corpo del documento	17
3.1.4. Verbale	17
3.1.4.1. Corpo del verbale	17
3.1.4.2. Verbale interno	17
3.1.4.3. Verbale esterno	17

3.1.5.	Versionamento	17
3.1.6.	Nomenclatura	18
3.1.7.	Regole stilistiche	18
3.1.8.	Strumenti	18
3.2.	Verifica	18
3.2.1.	Scopo	18
3.2.2.	Descrizione	19
3.2.3.	Analisi statica	19
3.2.4.	Analisi dinamica	19
3.2.4.1.	Test di unità	19
3.2.4.2.	Test di integrazione	19
3.2.4.3.	Test di sistema	20
3.2.4.4.	Test di regressione	20
3.2.4.5.	Test di accettazione	20
3.2.5.	Validazione	20
4.	Processi organizzativi	21
4.1.	Gestione organizzativa	21
4.1.1.	Scopo	21
4.1.2.	Ruoli	21
4.1.2.1.	Responsabile	21
4.1.2.2.	Amministratore	21
4.1.2.3.	Analista	21
4.1.2.4.	Progettista	22
4.1.2.5.	Programmatore	22
4.1.2.6.	Verificatore	22
4.1.3.	Gestione dei task e tracking delle attività	22
4.1.4.	Miglioramento	23
4.1.5.	Formazione	23
4.1.5.1.	Pianificazione	23
4.1.5.2.	Raccolta materiale	24
5.	Standard ISO/IEC 9126 per la qualità	25
5.1.	Funzionalità	25
5.2.	Affidabilità	25
5.3.	Usabilità	25
5.4.	Efficienza	26
5.5.	Manutenibilità	26
5.6.	Portabilità	26
6.	Metriche di qualità	28
6.1.	Metriche di qualità del processo	28
6.1.1.	Fornitura	28
6.1.2.	Sviluppo	28
6.1.3.	Documentazione	28
6.1.4.	Gestione della qualità	29
6.2.	Metriche di qualità del prodotto	29
6.2.1.	Funzionalità	29
6.2.2.	Affidabilità	29
6.2.3.	Manutenibilità	29
6.2.4.	Efficienza	30

1. Introduzione

1.1. Scopo 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_G*. Per gestire il *prodotto_G*, che comprende *software_G* e documentazione, è necessario adottare un approccio strutturato al ciclo di vita. Tale documento è redatto secondo lo standard *ISO12207:1995_G*, il quale identifica i processi di un *ciclo di vita_G* di un *software_G*, secondo una struttura modulare con relativa responsabilità su ciascun processo.

1.2. Scopo del progetto

Lo scopo del progetto è quello di sviluppare un *sistema_G* di testing e valutazione per misurare l'accuratezza, la coerenza e l'affidabilità delle risposte di un modello *AI_G* in risposta a domande specifiche. L'obiettivo è garantire che i modelli basati su Large Language Models (*LLM_G*) offrano prestazioni prevedibili e verificabili in contesti applicativi.

1.3. Glossario

Per evitare ambiguità o incomprensioni riguardanti la terminologia usata nel documento, è stato deciso di adottare un glossario in cui vengono riportate le varie definizioni. In questa maniera in esso verranno posti tutti i termini specifici del dominio d'uso con relativi significati. Tali termini sono marcati con una G a pedice, in *questo modo_G*.

1.4. Riferimenti

1.4.1. Riferimenti normativi

- **Regolamento del progetto didattico** (ultimo accesso: 2025-08-25):
<https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf>
- **Standard *ISO_G/IEC_G 12207:1995*** (ultimo accesso: 2025-08-25):
<https://www.math.unipd.it/~tullio/IS-1/2010/Approfondimenti/A03.pdf>

1.4.2. Riferimenti informativi

- **Capitolato C1** (ultimo accesso: 2025-08-25):
<https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C1.pdf>
- **T2 - Processi di ciclo di vita del software** (ultimo accesso: 2025-08-25):
<https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T02.pdf>
- Glossario v2.0.0:
<https://7commits.github.io/7Commits/glossary.html>

2. Processi primari

2.1. Fornitura

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_G* software richiesto dal *committente_G*.

2.1.1. Scopo

Il *processo_G* 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_G* finale rispetti i requisiti concordati con il *committente_G*, 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 del progetto.

2.1.2. Documentazione da consegnare

Di seguito vengono elencati i documenti che saranno consegnati all'azienda *Zucchetti* e ai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.

2.1.2.1. Analisi dei Requisiti

L'*Analisi dei Requisiti_G* è un documento che ha l'obiettivo di definire le *funzionalità_G* che il *prodotto_G* deve necessariamente avere per soddisfare a pieno le richieste del *Proponente_G*. Il documento di *Analisi dei Requisiti_G* è formato da una serie di definizioni essenziali:

- **Caso d'uso_G**: rappresentano in modo formale le *funzionalità_G* di un *sistema_G*, illustrando le *attività_G* svolte durante un'interazione;
- **UML_G Caso d'uso**: rappresentano in modo grafico/visivo l'interazione tra un *attore_G* e uno o più casi d'uso;
- **Requisiti**: l'insieme delle *funzionalità_G* richieste e quelle proposte in sede interna al gruppo.

2.1.2.2. Piano di Progetto

Il *Piano di Progetto_G* è un documento redatto dal responsabile con l'aiuto degli amministratori che ha come obiettivo quello di delineare la pianificazione e la gestione delle *attività_G* necessarie per la realizzazione del progetto. È composto dai seguenti punti:

- **Analisi dei rischi**;
- **Modello di sviluppo**;
- **Pianificazione**;
- **Preventivo**.

2.1.2.3. Piano di Qualifica

Il *Piano di Qualifica_G v2.0.0*, redatto dall'*amministratore_G*, descrive gli approcci e le strategie che il gruppo ha adottato per garantire la *qualità_G* del *prodotto_G*. Lo scopo di questo documento è quello di definire le modalità di verifica e validazione, oltre che gli standard e le procedure di *qualità_G* che il gruppo ha deciso di adottare per il ciclo di vita del progetto.

Si compone delle sezioni riguardanti:

- **Qualità di processo**: definizione dei parametri e delle metriche da rispettare per garantire processi di *qualità_G* elevata;

- **Qualità di prodotto:** definizione dei parametri e delle metriche da rispettare per garantire un *prodotto_G* finale di *qualità_G* elevata;
- **Test:** descrizione dei *test_G* necessari per assicurare che i requisiti stabiliti vengano soddisfatti nel *prodotto_G*
- **Valutazioni per il miglioramento:** resoconto dell'*attività_G* di verifica svolta e delle criticità riscontrate durante il *processo_G* di sviluppo del *software_G*.

2.1.2.4. Glossario

Il *Glossario v2.0.0* serve come un catalogo completo dei termini tecnici impiegati all'interno del progetto, fornendo definizioni chiare e precise. L'obiettivo di questo documento previene fraintendimenti a favore di una comprensione condivisa della terminologia specifica, migliorando la coerenza e la *qualità_G* della documentazione prodotta dal gruppo.

2.2. Sviluppo

Il *processo_G* di sviluppo rappresenta la serie di *attività_G* svolte dal team 7Commits al fine di implementare il *prodotto_G* software, rispettando le scadenze e i requisiti concordati col *Proponente_G*. Il *processo_G* è suddiviso nelle seguenti *attività_G*:

- *Analisi dei Requisiti_G*
- Progettazione;
- Codifica;
- Testing;
- Integrazione software.

2.2.1. Analisi dei requisiti

Lo scopo dell'*Analisi dei Requisiti_G* è comprendere e definire in modo chiaro e completo le necessità e le aspettative del *Proponente_G* e degli utenti relativamente al *prodotto_G* software.

2.2.1.1. Implementazione

L'analisi dei requisiti, raccolta nel documento *Analisi_dei_Requisiti_v1.0.0*, viene svolta secondo le seguenti fasi:

- Studio del *Capitolato_G* e delle esigenze del *Proponente_G*
- Individuazione dei casi d'uso e dei requisiti;
- Confronto con il *Proponente_G* su quanto *prodotto_G*
- Divisione dei requisiti nelle categorie individuate e applicazione dei quanto emerso nella discussione col *Proponente_G*.

L'*attività_G* di analisi può essere svolta in modo incrementale, quindi le sue fasi possono essere svolte più volte durante lo sviluppo del progetto.

2.2.1.2. Casi d'uso

I casi d'uso sono strutturati nel seguente modo:

- **Identificazione:** identificativo numerico del *caso d'uso_G*. Definita nel seguente formato:

UC<ID use case>:<nome caso d'uso>

- **Attore:** l'*attore_G* che intende compiere lo scopo rappresentato dal caso d'uso;
 1. Attore primario: è l'*attore_G* che innesca il caso d'uso per ottenere un risultato di valore.
 2. Attore secondario: è un *attore_G* che partecipa al caso d'uso, ma non ne è il protagonista principale. Fornisce supporto o riceve effetti indiretti.
- **Precondizioni:** stato in cui il *sistema_G* si deve trovare prima dell'avvio della funzionalità rappresentata dal caso d'uso;
- **Postcondizioni:** stato in cui il *sistema_G* si troverà dopo che l'utente avrà terminato lo scopo rappresentato dal caso d'uso;
- **Scenario_G principale:** descrizione della funzionalità rappresentata dal caso d'uso;
- **Generalizzazioni** (se necessario);
- **Estensioni** (se presenti);
- **Specializzazioni** (se presenti).

Di seguito è riportato un template del *caso d'uso_G*:

<Identificativo caso d'uso>:<descrizione caso d'uso>	
Attore principale	<attore principale>
Attori secondari	<lista attori secondari, se esistono>
Descrizione	<breve descrizione caso d'uso>
Pre-condizioni	<descrizione pre-condizione>
Post-condizioni	<descrizione post-condizione>
Scenario principale	1. <descrizione 1 azione> 2. <descrizione 2 azione>
Generalizzazioni	1. <UCx.x>:<descrizione 1 generalizzazione> 2. <UCx.x>:<descrizione 2 generalizzazione>

Tabella 1: Template caso d'uso

2.2.1.3. Diagramma casi d'uso

Un diagramma dei casi d'uso è uno strumento di modellazione che rappresenta visivamente le funzionalità di un *sistema_G* e le modalità con cui gli utenti interagiscono con esso. È particolarmente utile nella progettazione di sistemi poiché offre una rappresentazione intuitiva delle dinamiche operative e delle interazioni tra attori e *sistema_G*, senza entrare nei dettagli implementativi. I componenti principali di un diagramma dei casi d'uso sono:

- **Sistema:** delimita i confini del *sistema_G*, indicano quali funzionalità sono incluse e quali ne sono esterne. Il *sistema_G* viene rappresentata nei diagrammi come mostrato in Figura 1.

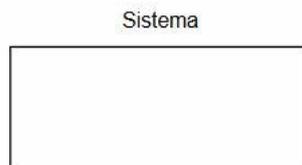


Figura 1: Rappresentazione sistema

- **Attori:** gli attori rappresentano entità esterne (umane o meno) che interagiscono con il *sistema_G* e sono raffigurati con un'icona stilizzata e un'etichetta identificativa. Possono essere generalizzati: un *attore_G* generico può avere attori più specifici che ne ereditano le funzionalità e aggiungono comportamenti contestuali;



Figura 2: Rappresentazione attore

- **Casi d'uso:** un *caso d'uso_G* descrive un'operazione che un utente può compiere attraverso il *sistema_G*. Ogni caso d'uso ha un'identificazione univoca e una breve descrizione della funzione. Può includere sequenze di azioni che illustrano le possibili interazioni con il *sistema_G* ed è collegato agli attori autorizzati tramite linee continue.



Figura 3: Rappresentazione caso d'uso

Nei diagrammi in questione poi possono comparire delle relazioni:

- **Associazione:** è la relazione fondamentale che collega un *attore_G* a un caso d'uso al quale partecipa. Uno stesso *attore_G* può partecipare a più casi d'uso.

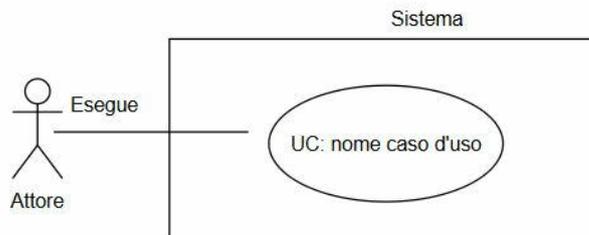


Figura 4: Rappresentazione associazione

- **Generalizzazioni:** le generalizzazioni possono riguardare sia gli attori che i casi d'uso. Gli attori o i casi figli ereditano le funzionalità dei genitori, aggiungendo aspetti specifici. La relazione è rappresentata con una freccia continua e un triangolo vuoto bianco;

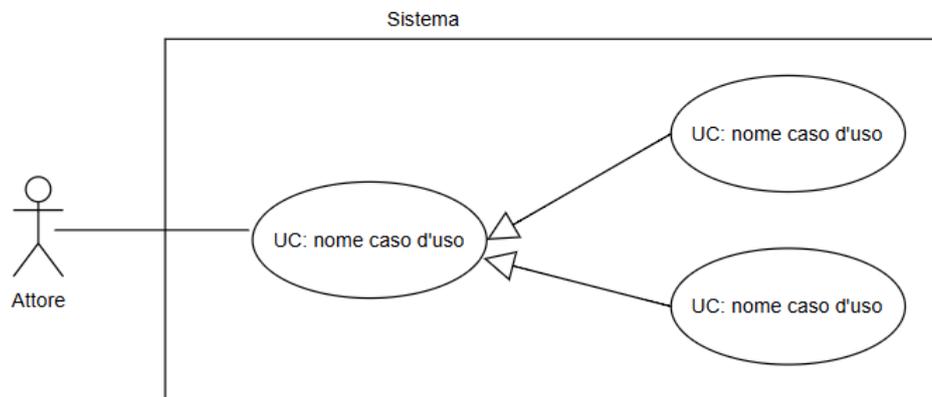


Figura 5: Rappresentazione generalizzazione tra casi d'uso

- **Inclusioni:** si verificano quando un caso d'uso ne richiama un altro in modo obbligatorio. Questo favorisce la riduzione della duplicazione e il riutilizzo delle strutture. La relazione è indicata con una freccia tratteggiata e l'etichetta "include";

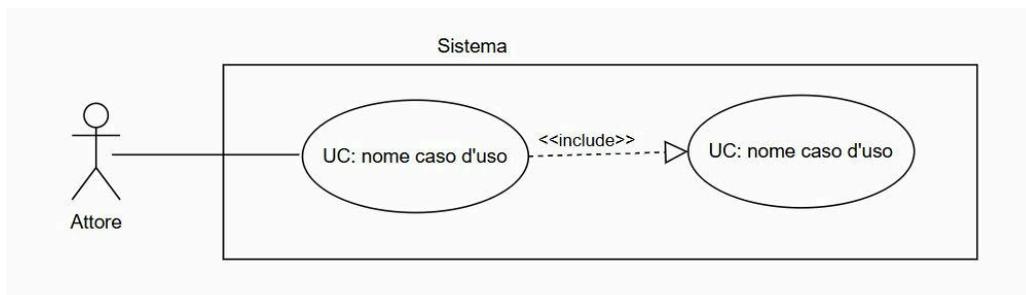


Figura 6: Rappresentazione inclusione

- **Estensioni:** rappresentano relazioni condizionali in cui un caso d'uso aggiuntivo viene eseguito solo in circostanze particolari, interrompendo temporaneamente il flusso principale. La relazione è raffigurata con una freccia tratteggiata e l'etichetta "extend".

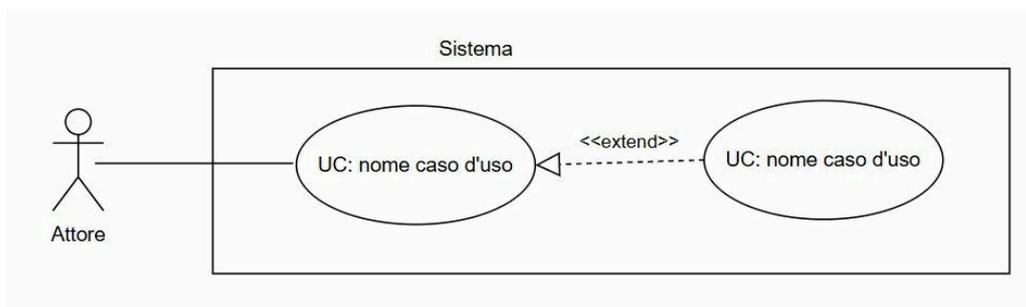


Figura 7: Rappresentazione estensione

2.2.1.4. Notazione dei casi d'uso

I casi d'uso seguono la seguente notazione: **UC**[Codice] : [Titolo] dove:

- **UC** sta per *Use Case*;
- [Codice] è l'identificativo univoco del caso d'uso. Si tratta di un numero intero progressivo assegnato in base all'ordine di descrizione, se il caso d'uso non ha padre, altrimenti se si tratta di un sottocaso d'uso si segue la notazione [Codice padre].[Numero figlio], ricorsivamente senza porre limite alla profondità della gerarchia;
- [Titolo] è il titolo del caso d'uso.

2.2.1.5. Requisiti: Notazione

Ogni *requisito_G* analizzato sarà identificato univocamente da una sigla del tipo **R**[Tipo][Priorità][Codice] nella quale:

- **R** sta per Requisito;
- [Tipo] può essere:
 - **F** se Funzionale;
 - **Q** se di Qualità;
 - **V** se di *Vincolo_G*.
- [Priorità] può essere:
 - **O** per Obbligatorio;
 - **D** per Desiderabile;
 - **P** per Opzionale.
- [Codice]: identifica univocamente i requisiti per ogni tipologia. È un numero intero progressivo univoco assegnato in ordine di importanza se il *requisito_G* non ha padre, se invece si tratta di un sotto-requisito segue il formato [Codice padre].[Numero figlio] e trattandosi di una struttura ricorsiva non c'è limite alla profondità della gerarchia.

2.2.1.6. Requisiti: Suddivisione

1. **Requisiti Funzionali**: descrivono le funzionalità del *sistema_G*, le azioni che il *sistema_G* può compiere e le informazioni che il *sistema_G* può fornire. Seguendo la notazione sopra riportata, si possono partizionare in:

- **RFO-** : Requisito Funzionale Obbligatorio;
- **RFD-** : Requisito Funzionale Desiderabile;
- **RFP-** : Requisito Funzionale Opzionale.

2. **Requisiti di Qualità**: descrivono come un *sistema_G* deve essere, o come il *sistema_G* deve essere visualizzato, per soddisfare le esigenze dell'utente. Seguendo la notazione sopra riportata, si possono partizionare in:

- **RQO-** : Requisito di Qualità Obbligatorio;
- **RQD-** : Requisito di Qualità Desiderabile;
- **RQP-** : Requisito di Qualità Opzionale.

3. **Requisiti di *Vincolo_G***: descrivono i limiti e le restrizioni normative/legislative che un *sistema_G* deve rispettare per soddisfare le esigenze dell'utente. Seguendo la notazione sopra riportata, si possono partizionare in:

- **RVO-** : Requisito di *Vincolo_G* Obbligatorio;
- **RVD-** : Requisito di *Vincolo_G* Desiderabile;
- **RVP-** : Requisito di *Vincolo_G* Opzionale.

2.2.2. Progettazione

Lo scopo della progettazione è definire l'architettura del *prodotto_G software_G* e fornire i componenti e le interazioni del *sistema_G* per garantire che funzioni in modo efficiente ed efficace rispetto ai requisiti funzionali e non funzionali determinati durante l'*attività_G* di *Analisi dei Requisiti_G*.

2.2.2.1. Documentazione

La progettazione porterà alla redazione del documento di Specifica Tecnica. Questo documento ha principalmente lo scopo di descrivere l'architettura del *prodotto_G software_G* e di mettere a disposizione una linea guida per garantire che il *sistema_G* venga implementato secondo i requisiti del progetto. Gli argomenti trattati in questo documento sono:

- **Tecnologie**: espone un'analisi delle tecnologie e dei linguaggi di programmazione utilizzati, delle librerie e dei *framework_G* necessari, oltre che delle infrastrutture realizzate, riportando in particolare vantaggi e svantaggi di ognuna;
- **Architettura di sistema**: descrive la struttura generale del *software_G*, la suddivisione in moduli o livelli e le interazioni tra i componenti;
- **Architettura delle componenti**: fornisce i dettagli dei singoli moduli o componenti del *sistema_G*, descrivendone responsabilità, interfacce, flussi di dati, modalità di interazione ed esplicitando eventuali sottocomponenti;
- **Progettazione di dettaglio**: definisce nel dettaglio gli algoritmi, le strutture dati, i flussi operativi e l'implementazione dei singoli componenti.

2.2.2.2. Fasi della progettazione

2.2.2.3. Progettazione logica

La *fase_G* di progettazione logica consiste nel definire un'architettura che illustra la struttura e il comportamento del *sistema_G* a un livello astratto, senza preoccuparsi delle tecnologie specifiche. In questa *fase_G* bisogna assicurarsi che i componenti siano descritti in modo dettagliato e che coprano tutti i requisiti funzionali e non, semplificando così la prossima *fase_G* della progettazione.

2.2.2.4. Progettazione di dettaglio

La *fase_G* di progettazione di dettaglio si concentra sulla definizione precisa di tutti gli aspetti tecnici del *sistema_G*, traducendo le specifiche logiche in soluzioni concrete e implementabili. In questa *fase_G*, vengono progettati i singoli componenti del *sistema_G*, insieme alle loro interfacce, strutture dati, classi e funzioni, e la comunicazione tra i moduli. L'intento è sviluppare una base solida e completa per l'implementazione del *software_G*, con chiarezza e precisione su tutte le decisioni progettuali.

2.2.2.5. Diagrammi delle classi

I diagrammi delle classi contenuti nel documento di Specifica Tecnica devono rispettare la notazione e le specifiche dalla versione 2.5 di *UML_G*.

2.2.2.6. Strumenti

Gli strumenti utilizzati per il *processo_G* di progettazione sono:

- Draw.io;
- *Typst_G*
- *GitHub_G*
- *Streamlit_G*.

2.2.3. Codifica

Lo scopo del *processo_G* di codifica è trasformare la progettazione dettagliata in codice sorgente eseguibile, utilizzando un linguaggio di programmazione appropriato. Questa fase consiste nello sviluppo effettivo delle funzionalità del *sistema_G*, garantendo che il codice sia corretto, efficiente, manutenibile e conforme agli standard di *qualità_G* e sicurezza. Inoltre, la codifica deve seguire le specifiche definite nelle fasi precedenti, assicurando che il *software_G* soddisfi i requisiti funzionali e non funzionali. Questa sezione ha lo scopo di normare la scrittura del codice al fine di:

- Rendere il codice più leggibile, uniforme e robusto;
- Semplificare e velocizzare il *processo_G* di verifica;
- Agevolare la *manutenzione_G*, il debugging e l'estensione del *software_G*
- Migliorare la *qualità_G* complessiva del codice.

2.2.4. Testing

La progettazione porterà alla redazione del documento di *Specifica Tecnica_G*. Questo documento ha lo scopo di descrivere l'architettura del *prodotto_G software_G* e di mettere a disposizione una linea guida per garantire che il *sistema_G* venga implementato secondo i requisiti del progetto. Gli argomenti trattati in questo documento sono:

- **Tecnologie:** espone un'analisi delle tecnologie e dei linguaggi di programmazione utilizzati, delle librerie e dei *framework_G* necessari, oltre che delle infrastrutture realizzate, riportando in particolare vantaggi e svantaggi di ognuna;
- **Architettura di *sistema_G*:** descrive la struttura generale del *software_G*, la suddivisione in moduli o livelli e le interazioni tra i componenti;
- **Architettura delle componenti:** fornisce i dettagli dei singoli moduli o componenti del *sistema_G*, descrivendone responsabilità, interfacce, flussi di dati, modalità di interazione ed esplicitando eventuali sottocomponenti;
- **Progettazione di dettaglio:** definisce nel dettaglio gli algoritmi, le strutture dati, i flussi operativi e l'implementazione dei singoli componenti.

2.2.4.1. Tipologie di test

I test realizzabili possono essere suddivisi in tre categorie principali:

- **Test di unità:** verificano il corretto funzionamento di una singola unità di codice indipendente (ad esempio una funzione), assicurandosi che produca i risultati attesi al variare dei possibili input, e vengono generalmente automatizzati per facilitare l'individuazione degli errori durante la fase di sviluppo;

- **Test di integrazione:** verificano il corretto funzionamento delle interazioni tra diverse unità di codice o componenti di un *sistema_G*, assicurandosi che, una volta integrati, i vari moduli lavorino insieme senza problemi, rilevando eventuali errori nelle interfacce e nei flussi di dati tra di essi;
- **Test di *sistema_G*:** verificano il comportamento complessivo di un'intera applicazione o *sistema_G*, testando tutte le sue componenti integrate per assicurarsi che soddisfi i requisiti funzionali e non funzionali, assicurandosi di valutare il *sistema_G* nel suo insieme simulando l'uso reale per identificare eventuali problemi di performance, sicurezza o compatibilità;

2.2.4.2. Notazione dei test

È stata decisa come notazione per identificare univocamente i test la seguente:

T[Tipologia][Numero]

Tipologia indica la tipologia del test:

- U: di unità;
- I: di integrazione;
- S: di *sistema_G*

Ogni test si trova in uno Stato, che può essere:

- V: verificato. Questo stato indica che il test ha fornito un esito positivo;
- NV: non verificato. Questo stato indica che il test ha fornito un esito negativo;
- NI: non implementato. Questo stato indica che il test non è ancora stato implementato, e quindi non fornisce nessun esito.

2.2.4.3. Strumenti

Gli strumenti utilizzati per il *processo_G* di testing del codice sono:

- Visual Studio Code;
- *GitHub_G*.

3. Processi di supporto

3.1. Documentazione

3.1.1. Scopo

Il *processo_G* di documentazione serve a tenere traccia di tutti i processi e *attività_G* relativi al *ciclo di vita del Software_G*, riportando le decisioni adottate e le *norme_G* attuate dal gruppo durante lo svolgimento del progetto. Le *norme_G* stabilite all'interno di questo documento verranno rispettate da tutti i membri del gruppo  7Commits.

3.1.2. Strumento utilizzato

Per la redazione dei documenti del progetto, utilizziamo *Typst_G*, una tecnologia avanzata e versatile per la scrittura e la formattazione di testi. Grazie alla sua sintassi intuitiva e alla capacità di generare documenti di alta *qualità_G*, *Typst_G* permette di combinare *efficienza_G* e personalizzazione. Questo strumento garantisce una gestione semplificata del layout e uno stile professionale, ideale per soddisfare le esigenze di documentazione tecnica e collaborativa.

3.1.3. Struttura dei documenti

Ogni documento prevede una struttura base per garantire la coerenza tra i vari documenti composta da queste sezioni:

- Pagina di copertina;
- Registro delle modifiche;
- Indice;
- Corpo del documento.

3.1.3.1. Pagina di copertina

Ogni prima pagina del documento, a prescindere dalla tipologia dello stesso, deve contenere le seguenti parti:

- **Titolo del documento;**
- **Versione del documento:** fare riferimento alla Sezione 3.1.5;
- **Logo del gruppo;**
- **Email del gruppo:** 7commits@gmail.com ;
- **Tabella con destinatari, redattori e verificatori.**

3.1.3.2. Registro delle modifiche

La seconda pagina è destinata a tenere traccia delle modifiche avvenute nel documento in modalità *LIFO_G*, ovvero deve contenere i numeri di versione dal maggiore al minore, rispecchiando le modifiche al documento in ordine cronologico inverso: dalla più recente alla più datata. La tabella è organizzata nel seguente modo:

- **Versione:** fare riferimento alla Sezione 3.1.5;
- **Data:** nella forma Anno-Mese-Giorno;
- **Autori:** chi ha effettuato la modifica al documento;
- **Verificatori:** chi ne ha verificato la modifica;

- **Descrizione delle modifiche:** un indicazione sintetica delle modifiche apportate al documento.

3.1.3.3. Indice

Nella pagina successiva al registro delle modifiche è presente l'indice del documento che riporta tutte le sezioni del documento

3.1.3.4. Corpo del documento

Il contenuto del documento è organizzato in capitoli che possono contenere sezioni e sottosezioni in base alle esigenze.

3.1.4. Verbale

3.1.4.1. Corpo del verbale

Nel caso in cui si stia redigendo un verbale il corpo del verbale deve essere così organizzato:

- **Informazioni sulla riunione:** contiene il luogo o la piattaforma e la data e l'ora nella quale il gruppo si è riunito assieme a una tabella sulla quale sono riportate le presenze dei vari membri;
- **Ordine del giorno:** i punti trattati nella riunione;
- **Sintesi dell'incontro:** una descrizione breve di cosa si è discusso.

3.1.4.2. Verbale interno

Nel *verbale interno*_G dovranno essere inclusi i seguenti punti:

- **Decisioni prese;**
- **Tabella attività:** organizzata in 3 colonne:
 - *ID* per riferimento alla *issue*_G su *GitHub*_G
 - *Dettaglio* per una descrizione breve della *issue*_G
 - *Assegnato/i* per indicare a chi è stata assegnata quella *attività*_G.

Nel caso la riunione coincida o sia successiva ad una *Sprint Review*_G, viene aggiunta una sezione:

- **Ruoli Sprint:** contiene una tabella con i nuovi ruoli per lo *Sprint*_G successivo.

3.1.4.3. Verbale esterno

Nel *verbale esterno*_G dovranno essere inclusi i seguenti punti:

- **Domande effettuate con relative risposte;**
- **Spazio per l'approvazione dell'azienda:** in basso a destra per permettere l'apposizione della firma del documento da parte dell'azienda.

3.1.5. Versionamento

Il *versionamento*_G dei documenti è organizzato nella forma **x.y.z**:

- **x:** indica la versione definitiva e approvata del documento;
- **y:** viene incrementato in seguito a modifiche emerse durante una verifica;
- **z:** viene incrementato ad ogni modifica.

3.1.6. Nomenclatura

I documenti saranno così denominati:

- Per i verbali verrà adottata la seguente nomenclatura: [VI_AAAA_MM_GG_vx.y.z] oppure [VE_AAAA_MM_GG_vx.y.z], dove:
 - VI indica un verbale interno, VE un verbale esterno;
 - AAAA_MM_GG rappresenta la data del verbale nel formato anno_mese_giorno;
 - vx.y.z corrisponde alla versione del documento secondo le regole descritte nella Sezione 3.1.5.
- Per gli altri documenti si seguiranno le seguenti regole: [nome_del_documento] con ogni parola separata da un «_» seguito dal numero di versione [vx.y.z], per esempio *Norme_di_progetto_v2.0.0*.

3.1.7. Regole stilistiche

Vengono utilizzate le seguenti regole stilistiche:

- I titoli delle sezioni iniziano con la lettera maiuscola;
- Ciascuna voce di un elenco deve essere seguita da un punto e virgola, ad eccezione dell'ultima che deve terminare con un punto;
- Le date vengono scritte nel formato YYYY-MM-DD (anno-mese-giorno).

Si utilizzerà il grassetto per evidenziare:

- Titoli delle sezioni;
- Parole di rilievo negli elenchi puntati.

Si utilizzerà il «[blu](#)» per indicare:

- Link;
- Termini del Glossario.

3.1.8. Strumenti

Gli strumenti utilizzati durante tutta la realizzazione della documentazione sono:

- *GitHub*_G: piattaforma di *hosting*_G per progetti *software*_G
- *Typst*_G: linguaggio per la stesura dei documenti, tramite *typst.app*.

3.2. Verifica

3.2.1. Scopo

La verifica rappresenta una *fase*_G fondamentale del ciclo di sviluppo del *software*_G, poiché consente di verificare che ogni risultato *prodotto*_G sia conforme ai requisiti stabiliti. Si tratta di un *processo*_G sistematico e continuo, che accompagna lo sviluppo sin dalle fasi iniziali, con l'obiettivo di individuare eventuali errori. Ogni risultato *prodotto*_G viene quindi esaminato secondo criteri predefiniti, per assicurare che rispecchi pienamente le aspettative progettuali. Questo *processo*_G non solo favorisce un'elevata qualità del software, ma crea anche le condizioni ottimali per una successiva e solida *fase*_G di validazione, facilitando la transizione verso la corretta conclusione del progetto.

3.2.2. Descrizione

Come descritto nel paragrafo precedente, la verifica è un'*attività_G* fondamentale nel ciclo di vita di un software e viene svolta in più fasi e in modo continuativo durante il progetto. A occuparsene è un team di verificatori dedicato, distinto da chi ha partecipato allo sviluppo, per garantire imparzialità e obiettività nell'analisi. La verifica coinvolge tutti gli elementi prodotti (codice, documentazione, report, ecc.), valutandoli in base a criteri di correttezza, completezza e coerenza. Elemento centrale di questo *processo_G* è il *Piano di Qualifica_G*, un documento guida che definisce per iscritto tutte le fasi della verifica. Le evidenze raccolte vengono accuratamente documentate, assicurando trasparenza, tracciabilità e la possibilità di ripetere il *processo_G* in modo consistente.

3.2.3. Analisi statica

L'analisi statica è una tecnica di verifica che si svolge senza l'esecuzione del codice, focalizzandosi solo sull'esame diretto di codice sorgente, documentazione e altri prodotti del progetto. Il suo scopo è individuare tempestivamente errori o incongruenze, garantendo che il *prodotto_G* rispetti i requisiti formali e qualitativi sin dalle prime fasi dello sviluppo. Questa tipologia di analisi si basa su metodi di revisione manuale tra l'autore e il verificatore (*Walkthrough_G*), con anche checklist e linee guida (*Inspection_G*) per condurre un controllo approfondito in modo più rapido ed efficiente. Questa tecnica è particolarmente utile perché permette di correggere eventuali difetti quando i documenti sono ancora facilmente modificabili, contribuendo a migliorare la *qualità_G* generale del *prodotto_G* fin dall'inizio.

3.2.4. Analisi dinamica

L'analisi dinamica, invece, è una tecnica di verifica che si basa sull'esecuzione del *software_G* per valutarne il comportamento e identificare eventuali difetti. Consente quindi di osservare il *software_G* in azione, offrendo così una visione concreta della sua affidabilità. Elemento centrale dell'analisi dinamica sono i *test_G*, ovvero l'esecuzione controllata del codice utilizzando specifici dati di input, con l'obiettivo di verificare che il comportamento ottenuto corrisponda a quello atteso. Per essere efficace, un *test_G* deve essere decidibile, cioè produrre sempre lo stesso risultato a parità di condizioni, e ripetibile, ossia eseguibile più volte senza che l'esito venga alterato da fattori esterni. La *qualità_G* dell'analisi dinamica dipende da diversi fattori: la chiarezza dei requisiti, la *qualità_G* del codice sottoposto a *test_G* e la definizione accurata dei casi di *test_G*. Quest'ultimo viene elaborato dal verificatore, tenendo conto della complessità e delle funzionalità del software. L'automazione gioca un *ruolo_G* chiave in questa *fase_G*, grazie all'uso di strumenti specifici che guidano l'esecuzione dei *test_G*, simulano componenti non ancora sviluppate o non coinvolte direttamente e registrano e analizzano i risultati ottenuti. Esistono varie tipologie di *test_G* impiegate nell'analisi dinamica, ciascuna focalizzata su un livello diverso del *prodotto_G*. L'analisi dinamica, insieme a quella statica, contribuisce a garantire una verifica completa e approfondita della qualità del software.

3.2.4.1. Test di unità

I *test_G* di unità sono definiti sulle componenti *software_G* autonome più piccole e sono implementati principalmente durante la progettazione. Assumono due connotazioni diverse, in base al tipo di controllo che si vuole effettuare:

- **Test funzionali:** verificano che l'output effettivo corrisponda con il valore atteso;
- **Test strutturali:** verificano tutti i possibili cammini del codice tramite una serie di test.

3.2.4.2. Test di integrazione

Sono concepiti durante la *fase_G* di progettazione architetturale, successivamente ai test di unità. Verificano, con un approccio incrementale, la corretta integrazione tra le diverse unità *software_G* già testate. In aggiunta, è possibile annullare tali modifiche in modo da ripristinare uno stato sicuro nel

caso si verifichino errori durante l'esecuzione di questo genere di test. E' possibile seguire due approcci di integrazione:

- **Top-down:** si comincia con le componenti di *sistema_G* che hanno più dipendenze e maggiore rilevanza esterna, garantendo così la disponibilità immediata delle *funzionalità_G* di alto livello. Questo approccio consente di testare in modo più approfondito le *funzionalità_G* centrali, rendendole disponibili prioritariamente.
- **Bottom-up:** si inizia dalle componenti di *sistema_G* con meno dipendenze e maggior valore interno, ovvero quelle meno visibili all'utente.

3.2.4.3. Test di sistema

Sono concepiti successivamente ai *test_G* di integrazione, durante la stesura dell'*Analisi dei Requisiti_G*. Servono a garantire il corretto funzionamento del *sistema_G*. Verificano, in particolare, che tutti i requisiti software specificati nel *Capitolato_G* siano presenti e funzionanti.

3.2.4.4. Test di regressione

I controlli di regressione si assicurano che le correzioni o le espansioni apportate a specifiche componenti architettoniche non arrechino danni al resto del *sistema_G*. Questi controlli prevedono la ripetizione mirata di *test_G* di unità, integrazione e *sistema_G* essenziali per garantire che le modifiche non compromettano le *funzionalità_G* precedentemente verificate, evitando così il verificarsi di regressioni.

3.2.4.5. Test di accettazione

I *test_G* di accettazione servono a verificare che il *prodotto_G* finale soddisfi tutti i requisiti richiesti dal *committente_G*: per questo motivo devono essere svolti obbligatoriamente in sua presenza.

3.2.5. Validazione

La validazione costituisce l'essenziale verifica che il *prodotto_G* software sia in linea con i requisiti e le aspettative del cliente, rappresentando una *fase_G* cruciale nello sviluppo del *software_G*. Questo *processo_G* si concentra attentamente su diversi aspetti:

- **Conformità ai requisiti:** il *prodotto_G* deve soddisfare integralmente tutti i requisiti specificati dal cliente;
- **Funzionamento corretto:** il *prodotto_G* deve operare correttamente, in conformità con la logica di progettazione;
- **Usabilità:** il *prodotto_G* deve essere intuitivo e di facile utilizzo;
- **Efficacia_G:** il *prodotto_G* deve dimostrarsi efficace nel soddisfare le necessità dei clienti.

L'aspettativa finale è di giungere ad un *prodotto_G* che risponda pienamente ai requisiti identificati ed operi correttamente.

4. Processi organizzativi

4.1. Gestione organizzativa

4.1.1. Scopo

Questo *processo_G* ha l'obiettivo di formalizzare le modalità operative adottate dal gruppo per assicurare un efficace coordinamento delle *attività_G* progettuali. Vengono descritte le strategie di comunicazione, sia all'interno del team sia verso l'esterno, insieme agli strumenti impiegati per supportarle. Il documento definisce inoltre i criteri per l'assegnazione dei ruoli e delle responsabilità, nonché le *attività_G* e i compiti che ogni membro del gruppo è chiamato a svolgere, al fine di garantire ordine ed *efficienza_G* nel *processo_G* di sviluppo.

4.1.2. Ruoli

Per lo svolgimento di tutto il progetto sono stati individuati vari ruoli che a turno verranno ricoperti da ciascun membro del gruppo, ognuno per le mansioni, le responsabilità e le ore previste richieste.

4.1.2.1. Responsabile

Il Responsabile è la figura centrale che con le sue competenze tecniche ha il *compito_G* di coordinare in modo efficace le *attività_G* del gruppo, assicurando che il progetto proceda secondo quanto pianificato. Inoltre è il punto di riferimento per le comunicazioni con il *committente_G* e l'azienda *proponente_G*. Ha il *compito_G* di coordinare in modo efficace le *attività_G* del team, assicurando che il progetto proceda secondo quanto pianificato. Si occupa quindi di:

- Coordinare le *attività_G* tra il gruppo e gli enti esterni;
- Gestire le comunicazioni interne;
- Supervisionare la pianificazione del progetto, definendo *attività_G*, priorità e risorse;
- Assegnare task e monitorarne l'avanzamento;
- Approvare la documentazione prodotta;
- Analizzare e gestire i potenziali rischi;

4.1.2.2. Amministratore

L'*Amministratore_G* è colui che deve gestire le risorse umane, materiali, economiche e strutturali, quindi deve assicurarsi che queste siano presenti durante tutte le fasi di svolgimento del progetto. Deve garantire che tutte queste risorse siano combinate insieme in modo funzionale.

Riassunto deve occuparsi di:

- Approvvigionamento delle risorse;
- Garantire continuità e *funzionalità_G*
- Gestire eventuali errori;
- Mantenere viva la documentazione.

4.1.2.3. Analista

L'*Analista_G* deve analizzare con attenzione il *Capitolato_G* per identificare e formalizzare in modo accurato i requisiti del *sistema_G*. La sua *attività_G* è cruciale nelle fasi iniziali del progetto, poiché una corretta analisi rappresenta la base su cui costruire un progetto solido ed efficace. Sulla base di questo deve anche redigere il documento *Analisi dei Requisiti_G*, riferimento costante per le fasi successive.

Le *attività_G* fondamentali che deve svolgere sono quindi:

- Studiare il problema e il contesto;
- Raccogliere, interpretare e sintetizzare le esigenze espresse dai committenti;
- Redigere il documento *Analisi dei Requisiti_G*.

4.1.2.4. Progettista

Il Progettista è responsabile delle scelte realizzative del progetto, quindi si tratta di un esperto professionale, tecnico e tecnologico. Partendo dal lavoro svolto dall'*Analista_G*, modella il *sistema_G* per rispondere in modo efficace al problema, definendo le specifiche tecniche che guideranno lo sviluppo. Le sue principali responsabilità includono quindi:

- Definizione dell'architettura del *sistema_G* e scelte progettuali;
- Garantire che il *prodotto_G* finale sia economico, scalabile e facilmente manutenibile;
- Favorire l'*efficienza_G* e l'*efficacia_G* complessiva tramite decisioni tecniche consapevoli.

4.1.2.5. Programmatore

Il Programmatore è colui che si occupa della scrittura del codice, quindi di codifica del codice. Si occupa quindi di:

- Codificare ciò che è stato definito e scrivere codice;
- Eventuale futura *manutenzione_G* del codice;
- Redigere il *Manuale Utente_G*.

4.1.2.6. Verificatore

Il Verificatore ha il *compito_G* di controllare la *qualità_G* e la correttezza del lavoro svolto dagli altri membri del gruppo, garantendo che ogni *prodotto_G* rispetti gli standard tecnici e le *norme_G* di progetto e le aspettative qualitative condivise dal team. Trattandosi di un'*attività_G* trasversale a tutte le fasi dello sviluppo, questa figura è presente lungo l'intero ciclo di vita del progetto.

Le sue responsabilità principali comprendono quindi:

- Verificare che le *attività_G* svolte siano conformi alle *norme_G* di progetto e agli standard di *qualità_G* definiti;
- Utilizzare strumenti e tecniche di verifica per controllare la correttezza e la coerenza dei prodotti intermedi e finali;
- Segnalare eventuali non conformità e supportare il gruppo nell'individuazione delle correzioni necessarie.

4.1.3. Gestione dei task e tracking delle attività

Il gruppo  7Commits utilizza il *sistema_G* di Issue Tracking integrato in *GitHub_G* per la gestione operativa delle *attività_G*. Questo strumento consente un'organizzazione efficiente e trasparente dei compiti, con una chiara suddivisione delle responsabilità e un monitoraggio puntuale dello stato di avanzamento tramite le GitHub Projects *Board_G*.

È *compito_G* del Responsabile di progetto creare e assegnare le *Issue_G*, ognuna delle quali rappresenta un'unità di lavoro da svolgere. Le *Issue_G* vengono monitorate tramite la *Board_G* Kanban, che ne evidenzia visivamente lo stato: To Do, In Progress e Done.

Ogni *Issue_G* contiene:

- **Titolo:** descrizione sintetica e identificativa del *compito_G*

- **Descrizione (opzionale):** dettagli operativi dell'*attività_G* e riferimenti ai file/documenti coinvolti;
- **Assegnatario:** membro del gruppo incaricato dello svolgimento;
- **Milestone_G:** obiettivo o *fase_G* progettuale associata (Candidatura, RTB, PB, *CA_G*);
- **Etichette:** categorie che identificano la natura del *compito_G* (es. documentazione, verbali interni/esterni, bug);
- **Stato nella board di progetto:** *fase_G* corrente del task nella pipeline operativa (Todo, Done, In Progress).
- **Eventuali Parent Issue_G:** quando la issue si riferisce ad una issue principale come la scrittura di un documento gli viene assegnata una Parent *Issue_G*

La gestione di una nuova *attività_G* segue la seguente procedura:

- Il Responsabile apre una nuova *Issue_G* in stato To Do e assegna il task al membro designato;
- All'avvio dello sviluppo, l'assegnatario imposta lo stato su In Progress ;
- Al termine dell'*attività_G*, viene aperta una *pull request_G* collegata all'*Issue_G* tramite il comando `close #X` nella descrizione, dove X è il numero dell'*Issue_G*, e viene assegnato il Verificatore;

Il Verificatore esamina la *pull request_G*:

- Se l'esito è positivo:
 1. La *pull request_G* viene approvata e unita al *Branch_G* principale;
 2. L'*Issue_G* passa automaticamente allo stato Done.
- Se l'esito è negativo:
 1. Il verificatore fornisce feedback nella *Issue_G*
 2. L'incaricato apporta le correzioni richieste e ripresenta la *pull request_G* (si torna al punto 3).

Questo *processo_G* permette di mantenere traccia dell'*attività_G* svolta, assicurare la *qualità_G* del lavoro *prodotto_G* e garantire una gestione coordinata del progetto.

4.1.4. Miglioramento

Il *processo_G* di miglioramento mira a ottimizzare continuamente il *way of working_G*, aumentando l'*efficacia_G* e l'*efficienza_G* delle *attività_G* senza compromettere la *qualità_G*.

4.1.5. Formazione

Lo scopo del *processo_G* di formazione è aiutare i membri del gruppo a sviluppare le competenze necessarie per lavorare in modo efficace e produttivo. Questo *processo_G* si propone di garantire che le persone abbiano le conoscenze giuste per raggiungere gli obiettivi del progetto e rispettare gli standard di *qualità_G*.

4.1.5.1. Pianificazione

Ogni membro del team ha la libertà di scegliere come formarsi, seguendo un approccio pratico e personalizzato, in base alle proprie esigenze e interessi. Inoltre, chi ha più esperienza e competenze in un determinato campo è incoraggiato ad aiutare e supportare i colleghi con meno esperienza, creando un ambiente di apprendimento collaborativo. Questo permette a tutti di crescere insieme, condividendo conoscenze e risorse in modo continuo e dinamico.

4.1.5.2. Raccolta materiale

Il materiale per la formazione viene principalmente cercato e trovato online, sfruttando le risorse disponibili su internet. Ogni membro del team può accedere a tutorial, corsi, articoli, video e altre risorse gratuite per approfondire le proprie competenze in modo autonomo

5. Standard ISO/IEC 9126 per la qualità

La norma ISO/*IEC*_G 9126 è uno standard internazionale che ha contribuito a definire i parametri essenziali per valutare la qualità del software. Questa norma rappresenta un insieme di linee guida dettagliate e criteri di valutazione per gli attributi chiave della *qualità*_G del software. In particolare, identifica sei macro-categorie di attributi di *qualità*_G del software, ognuna delle quali è ulteriormente scomposta in sottoattributi specifici:

- **Funzionalità**
- **Affidabilità**
- **Usabilità**
- **Efficienza**
- **Manutenibilità**
- **Portabilità**

5.1. Funzionalità

La *funzionalità*_G si concentra sulla valutazione della capacità del *software*_G di fornire funzioni che soddisfano i requisiti specificati e impliciti. I suoi sottoattributi sono:

- **Adeguatezza:** é la capacità del *software*_G di fornire un insieme di *funzionalità*_G che soddisfano i requisiti specifici dell'utente e del contesto d'uso;
- **Accuratezza:** é la capacità del *software*_G di fornire risultati corretti con il grado di precisione richiesto;
- **Interoperabilità:** é la capacità del *software*_G di cooperare ed interagire efficacemente con altri sistemi;
- **Sicurezza:** é la capacità del *software*_G di proteggere i dati dagli accessi non autorizzati;
- **Conformità:** é la conformità del *software*_G agli standard, alle *norme*_G e alle specifiche funzionali pertinenti.

5.2. Affidabilità

L'affidabilità si concentra sulla valutazione della capacità del *software*_G di mantenere un adeguato livello di prestazioni anche in presenza di errori o malfunzionamenti. I suoi sottoattributi sono:

- **Maturità:** é la capacità del *software*_G di mantenere una stabilità durante l'esecuzione evitando errori e risultati non corretti;
- **Tolleranza agli errori:** é la capacità del *software*_G di gestire e tollerare errori;
- **Capacità di recupero:** é la capacità del *software*_G di ripristinare le prestazioni desiderate dopo che si è verificato un errore;
- **Affidabilità delle informazioni:** é la precisione e l'affidabilità delle informazioni fornite dal *software*_G.

5.3. Usabilità

L'usabilità è un aspetto critico nella valutazione della *qualità*_G del software, poiché influenza direttamente l'esperienza dell'utente durante l'interazione con il *sistema*_G. I suoi sottoattributi sono:

- **Comprensibilità:** é la facilità con cui l'utente può comprendere l'*interfaccia_G* utente, le informazioni presentate e il modo in cui eseguire le operazioni;
- **Apprendibilità:** é la facilità con cui nuovi utenti possono imparare ad utilizzare il *software_G*, attraverso una curva di apprendimento rapida e un accesso chiaro alle *funzionalità_G*
- **Operabilità:** é la facilità con cui gli utenti possono eseguire le operazioni desiderate in modo efficiente, senza errori e con un utilizzo ottimale delle risorse;
- **Attrattività:** é la piacevolezza estetica dell'*interfaccia_G* utente e del design complessivo del *software_G*
- **Aderenza all'usabilità:** si riferisce alla misura in cui il *software_G* rispetta le linee guida e le convenzioni dell'usabilità.

5.4. Efficienza

L'*efficienza_G* si concentra sulla capacità del *software_G* di utilizzare in modo ottimale le risorse a sua disposizione, garantendo prestazioni elevate e tempi di risposta rapidi. I suoi sottoattributi sono:

- **Utilizzo delle risorse:** misura quanto efficientemente il *software_G* sfrutti la capacità delle risorse di *sistema_G* durante l'esecuzione;
- **Aderenza all'efficienza:** indica quanto il *software_G* rispetti le linee guida e le migliori pratiche per l'ottimizzazione del codice e delle risorse;
- **Comportamento rispetto al tempo:** Rappresenta la capacità del *software_G* di rispondere rapidamente alle richieste degli utenti e di adattarsi dinamicamente alle variazioni di carico.

5.5. Manutenibilità

La manutenibilità si concentra sulla facilità con cui il *software_G* può essere modificato, corretto, adattato e migliorato nel tempo. I suoi sottoattributi sono:

- **Analizzabilità:** é la facilità con cui è possibile analizzare il codice sorgente per identificare errori, problemi di progettazione o aree di miglioramento;
- **Modificabilità:** é la facilità con cui il *software_G* può essere modificato senza causare effetti indesiderati o introduzione di nuovi errori;
- **Stabilità:** é la capacità del *software_G* di evitare effetti collaterali indesiderati durante le modifiche o l'introduzione di nuove *funzionalità_G*
- **Testabilità:** é la facilità con cui è possibile testare il *software_G* per garantire che le modifiche apportate non abbiano effetti indesiderati su *funzionalità_G* esistenti;
- **Aderenza alla manutenibilità:** si riferisce alla misura in cui il *software_G* rispetta le linee guida e le convenzioni della manutenibilità.

5.6. Portabilità

La portabilità si riferisce alla facilità con cui il *software_G* può essere trasferito da un ambiente a un altro senza dover apportare modifiche sostanziali. I suoi sottoattributi sono:

- **Adattabilità:** é la capacità del *software_G* di adattarsi a diversi ambienti senza richiedere modifiche sostanziali al codice sorgente o all'architettura;
- **Installabilità:** é la facilità con cui il *software_G* può essere installato in un nuovo ambiente;

- **Sostituibilità:** é la facilità con cui il *software_G* può sostituire o essere sostituito da altre applicazioni nello stesso ambiente;
- **Coesistenza:** é la capacità del *software_G* di operare in modo efficace e senza conflitti all'interno di un ambiente condiviso con altre applicazioni;
- **Aderenza alla portabilità:** si riferisce alla misura in cui il *software_G* rispetta le linee guida e le convenzioni della portabilità.

6. Metriche di qualità

6.1. Metriche di qualità del processo

6.1.1. Fornitura

- **CV (Cost Variance)**: Misura la deviazione dei costi rispetto al budget, se il costo è negativo significa che si è sforato il limite del budget (SPI > 1: in anticipo rispetto ai tempi pianificati, SPI < 1: in ritardo rispetto ai tempi pianificati). La formula è: $CV = EV - AC$.
- **PV (Planned Value)**: Il valore pianificato, ovvero il costo stimato del lavoro previsto in un determinato momento del progetto.
- **EV (Earned Value)**: Rappresenta il valore guadagnato, che rappresenta il costo stimato del lavoro effettivamente completato in quel momento.
- **AC (Actual Cost)**: Rappresenta il costo effettivo, cioè quanto è stato realmente speso fino a quel punto.
- **CPI (Cost Performance Index)**: Indica se il progetto sta spendendo meno o più del previsto (CPI > 1: sotto budget, CPI < 1: sopra budget). La formula è: $CPI = \frac{EV}{AC}$.
- **SPI (Schedule Performance Index)**: Rappresenta l'*efficienza_G* temporale con cui il lavoro pianificato è stato completato rispetto a quanto programmato. La formula è $SPI = \frac{EV}{PV}$.
- **BAC (*Budget At Completion_G*)**: Rappresenta il budget totale pianificato per il completamento del progetto.
- **EAC (Estimated At Completion)**: Rappresenta l'aggiornamento della stima del valore per la realizzazione del progetto, ovvero il BAC ricalcolato in base allo stato attuale del progetto. La formula è $EAC = \frac{BAC}{CPI}$.
- **VAC (Variance At Completion)**: Rappresenta la differenza tra il budget previsto e quello attuale alla fine del progetto. La formula è $VAC = BAC - EAC$.
- **ETC (Estimated To Completion)**: Rappresenta la valutazione del costo supplementare richiesto per portare a termine il progetto. La formula è $ETC = EAC - AC$.
- **SV (Schedule Variance)**: Indica se le *attività_G* pianificate del progetto sono in linea, anticipate o in ritardo rispetto alla programmazione. La formula è $SV = EV - PV$.
- **BV (Budget Variance)**: Indica se, alla data attuale, le spese sostenute sono superiori o inferiori rispetto a quanto originariamente previsto nel budget. La formula è $BV = PV - AC$.

6.1.2. Sviluppo

- **SC (Statement Coverage)**: Rappresenta la percentuale di istruzioni nel codice che vengono eseguite durante i test. La formula è $SC = \frac{N \text{ Statement eseguiti}}{N \text{ Statement totali}} * 100$.

6.1.3. Documentazione

- **IG (Indice Gulpease)**: Rappresenta un indicatore per analizzare la facilità di lettura di un testo scritto in italiano. L'Indice Gulpease si basa su due variabili linguistiche principali: la lunghezza delle parole e quella delle frasi. La formula per determinarlo è: $IG = 89 + \frac{(300 * NF - NL)}{NP}$, dove:
 - NF: Indica il numero delle frasi.
 - NL: Indica il numero delle lettere.
 - NP: Indica il numero di parole.

Questo indice fornisce un punteggio che varia da 0 a 100. I possibili punteggi possono essere:

- ▶ 0-55: Testo incomprensibile.
- ▶ 56-70: Testo molto difficile.
- ▶ 71-80: Testo difficile.
- ▶ 81-95: Testo facile.
- ▶ 95-100: Testo molto facile.

Per calcolarlo viene utilizzato un *software*_G online: https://farfalla-project.org/readability_static/

6.1.4. Gestione della qualità

- **MNS (Metriche Non Soddisfatte)**: Rappresenta le quantità di metriche che il progetto non riesce a soddisfare o mantenere.

6.2. Metriche di qualità del prodotto

6.2.1. Funzionalità

- **ROS (Requisiti Obbligatori Soddisfatti)**: Rappresenta la percentuale di requisiti obbligatori che sono stati soddisfatti durante la creazione del *prodotto*_G. La formula è:
$$\text{ROS} = \frac{\text{requisiti obbligatori soddisfatti}}{\text{requisiti obbligatori totali}} * 100.$$
- **RDS (Requisiti Desiderabili Soddisfatti)**: Rappresenta la percentuale di requisiti desiderabili che sono stati soddisfatti durante la creazione del *prodotto*_G. La formula è:
$$\text{RDS} = \frac{\text{requisiti desiderabili soddisfatti}}{\text{requisiti desiderabili totali}} * 100.$$
- **RPS (Requisiti Opzionali Soddisfatti)**: Rappresenta la percentuale di requisiti opzionali che sono stati soddisfatti durante la creazione del *prodotto*_G. La formula è:
$$\text{RPS} = \frac{\text{requisiti opzionali soddisfatti}}{\text{requisiti opzionali totali}} * 100.$$

6.2.2. Affidabilità

- **PTCP (Passed Test Cases Percentage)**: Rappresenta la percentuale di casi di *test*_G completati con successo rispetto al numero totale di casi di *test*_G pianificati. La formula è:
$$\text{PTCP} = \frac{\text{test superati}}{\text{test totali}} * 100.$$
- **CC (Code Coverage)**: Rappresenta il numero di linee di codice verificate con esito positivo all'interno di un *processo*_G di *test*_G. La formula è:
$$\text{CC} = \frac{\text{linee di codice scritte}}{\text{linee di codice totali}} * 100.$$

6.2.3. Manutenibilità

- **SFIN (Structure Fan IN)**: Rappresenta la quantità di moduli o componenti che interagiscono direttamente o dipendono da un modulo o una funzione specifica. Un valore elevato suggerisce che molte parti del *sistema*_G fanno affidamento su quel particolare modulo.
- **SFOUT (Structure Fan Out)**: Rappresenta la quantità di connessioni o relazioni che un componente o modulo ha con altri elementi del *sistema*_G. Questa misura riflette il numero di moduli che interagiscono o su cui si basa un determinato modulo. Un fan-out elevato può segnalare che un modulo è fortemente dipendente da molti altri.

6.2.4. Efficienza

- **TDE (Tempo di Elaborazione)**: Rappresenta il tempo di risposta dal momento in cui vengono inseriti dati all'interno del *prodotto* software al momento in cui vengono visualizzati dall'utente in questo caso l'installatore.