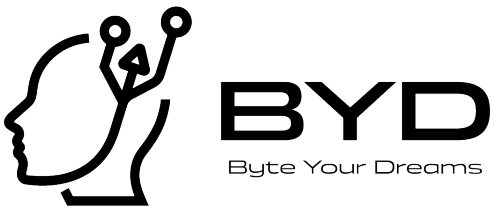

byteyourdreams.swe@gmail.com

Norme di progetto

Informazioni documento

Redattore	A.M. Margarit
Verificatore	A. Mio
Amministratore	L. Albertin
Destinatari	T. Vardanega R. Cardin

Registro delle modifiche

Versione	Data	Autore	Verificatore	Dettaglio
2.0.0	25/10/2025		L. Albertin	Approvazione documento
1.0.3	10/04/2025	L. Albertin	L. Zanesco	Aggiornamento struttura repository
1.0.2	20/03/2025	L. Albertin	L. Zanesco	Corretti riferimenti
1.0.1	10/03/2025	L. Albertin	L. Zanesco	Aggiornamento sez. Progettazione
1.0.0	09/02/2025		L. Albertin	Approvazione documento
0.6.2	07/02/2025	L. Albertin	L. Zanesco	Fix documento
0.6.1	14/01/2025	L. Albertin	L. Zanesco	Fix documento
0.6.0	14/01/2025	A.M Margarit	L. Zanesco	Redazione sez. Standard per la qualità
0.5.1	09/01/2025	L. Albertin	L. Zanesco	Fix documento
0.5.0	09/01/2025	A.M Margarit	L. Zanesco	Redazione sez. Processi organizzativi e Gestione della qualità
0.4.3	04/01/2025	A.M Margarit	A. Mio	Fix documento
0.5.0	05/12/2024	A. Mio	O.F. Stiglet	Redazione sez. Fornitura e Sviluppo
0.4.2	01/12/2024	L.Albertin	O.F. Stiglet	Fix documento
0.4.1	16/11/2024	L.Albertin	O.F. Stiglet	Fix introduzione
0.4.0	16/11/2024	L.Albertin	O.F. Stiglet	Redazione sez. Verifica
0.3.0	15/11/2024	L.Albertin	Y. Huang	Redazione sez. Gestione della configurazione
0.2.0	13/11/2024	L.Albertin	Y. Huang	Redazione sez. Documen-tazione
0.1.0	12/11/2024	A. Mio	L. Albertin	Redazione sez. Fornitura
0.0.1	10/11/2024	A.M. Margarit	A. Mio	Redazione

Indice

Byte Your Dreams

novembre 10, 2024

Contents

1 Introduzione **7**

1.1 Scopo del documento . . . 7

1.2 Glossario . . . 7

1.3 Riferimenti . . . 7

1.3.1 Riferimenti normativi . . . 7

1.3.2 Riferimenti informativi . . . 7

2 Processi primari **7**

2.1 Fornitura . . . 7

2.1.1 Introduzione . . . 7

2.1.2 *attività*$_G$ . . . 8

2.1.3 Comunicazioni con l'azienda proponente e documentazione fornita . . . 8

2.1.4 Documentazione fornita . . . 8

2.1.4.1 Valutazione dei capitolati . . . 8

2.1.4.2 Analisi dei requisiti v2.0.0 . . . 9

2.1.4.3 Piano di progetto v2.0.0 . . . 9

2.1.4.4 Piano di qualifica v2.0.0 . . . 9

2.1.4.5 Specifica tecnica v1.0.0 . . . 10

2.1.4.6 Manuale utente v1.0.0 . . . 10

2.1.4.7 Glossario v2.0.0 . . . 10

2.1.4.8 Lettera di presentazione . . . 10

2.1.5 Strumenti . . . 10

2.2 Sviluppo . . . 10

2.2.1 Introduzione . . . 10

2.2.2 Analisi dei requisiti . . . 11

2.2.2.1 Descrizione . . . 11

2.2.2.2 Obiettivi . . . 11

2.2.2.3 Casi d'uso . . . 11

2.2.2.4 Diagrammi dei casi d'uso . . . 12

2.2.2.5 Requisiti . . . 18

2.2.2.6 Metriche . . . 19

2.2.3 Progettazione . . . 20
2.2.3.1 Descrizione . . . 20
2.2.3.2 Obiettivi . . . 20
2.2.3.3 Documentazione . . . 22
2.2.3.4 Qualità dell'architettura . . . 22
2.2.3.5 Diagrammi UML . . . 23
2.2.3.6 Design pattern . . . 24
2.2.3.7 Strumenti . . . 24
2.2.4 Codifica . . . 24
2.2.4.1 Descrizione e Scopo . . . 24
2.2.4.2 Aspettative . . . 24
2.2.4.3 Norme di codifica . . . 25
2.2.5 Configurazione dell'ambiente di esecuzione . . . 25
2.2.5.1 Docker . . . 25
2.2.5.2 Strumenti . . . 25

3 Processi di supporto **26**
3.1 Documentazione . . . 26
3.1.1 Introduzione . . . 26
3.1.2 Ciclo di vita . . . 26
3.1.3 Sorgente documenti . . . 27
3.1.4 Nomenclatura dei documenti . . . 27
3.1.5 Struttura del documento . . . 27
3.1.5.1 Prima pagina . . . 27
3.1.5.2 Registro delle modifiche . . . 27
3.1.5.3 Indice . . . 28
3.1.5.4 Piè di pagina . . . 28
3.1.5.5 Verbali . . . 28
3.1.6 Regole stilistiche . . . 28
3.1.7 Strumenti . . . 29
3.2 Verifica . . . 29
3.2.1 Scopo . . . 29
3.2.2 Verifica dei documenti . . . 29
3.2.3 Analisi . . . 30
3.2.3.1 Analisi statica . . . 30
3.2.3.2 Analisi dinamica . . . 31
3.2.3.3 Test di unità . . . 31
3.2.3.4 Test di integrazione . . . 31
3.2.3.5 Test di Sistema . . . 31
3.2.3.6 Test di accettazione . . . 31
3.2.3.7 Sequenza delle fasi dei test . . . 31
3.2.4 Classificazione dei test . . . 31
3.2.5 Stato dei test . . . 32

3.2.6 Strumenti . . . 32
3.3 Validazione . . . 32
3.3.1 Introduzione . . . 32
3.3.2 Procedura di validazione . . . 32
3.3.3 Strumenti utilizzati . . . 33
3.4 Gestione della configurazione . . . 33
3.4.1 Introduzione . . . 33
3.4.2 Versionamento . . . 33
3.4.3 Repository . . . 33
3.4.3.1 Flusso generale Gitflow . . . 34
3.4.4 Struttura repository "Documents" . . . 35
3.4.5 Sincronizzazione e Branching . . . 35
3.4.5.1 Documentazione . . . 35
3.5 Risoluzione dei problemi . . . 36
3.5.1 Introduzione . . . 36
3.5.2 Gestione dei rischi . . . 36
3.5.2.1 Metriche . . . 36
3.6 Gestione della qualità . . . 37
3.6.1 Introduzione . . . 37
3.6.2 Attività . . . 37
3.6.3 Piano di qualifica . . . 38
3.6.4 PDCA . . . 38
3.6.5 Strumenti . . . 39
3.6.6 Struttura e identificazione metriche . . . 39
3.6.7 Criteri di accettazione . . . 39
3.6.8 Metriche . . . 39

4 Processi organizzativi **39**
4.1 Gestione dei Processi . . . 39
4.1.1 Introduzione . . . 39
4.1.1.1 Attività di gestione di processo . . . 40
4.1.2 Pianificazione . . . 40
4.1.2.1 Descrizione . . . 40
4.1.2.2 Obiettivi . . . 40
4.1.2.3 Assegnazione dei ruoli . . . 41
4.1.2.4 Responsabile . . . 41
4.1.2.5 Amministratore . . . 41
4.1.2.6 Analista . . . 41
4.1.2.7 Progettista . . . 41
4.1.2.8 Programmatore . . . 41
4.1.2.9 Verificatore . . . 41
4.1.2.10 Ticketing . . . 42
4.1.2.11 Strumenti . . . 43

4.1.3 Coordinamento . . . 43
4.1.3.1 Descrizione . . . 43
4.1.3.2 Obiettivi . . . 43
4.1.3.3 Comunicazioni sincrone . . . 43
4.1.3.4 Comunicazioni asincrone . . . 43
4.1.3.5 Riunioni interne . . . 43
4.1.3.6 Riunioni esterne . . . 44
4.1.3.7 Strumenti . . . 45
4.1.3.8 Metriche . . . 45

5 Standard per la qualità **45**
5.1 Caratteristiche del Sistema, Standard ISO/IEC 25010 . . . 45
5.1.1 Funzionalità . . . 45
5.1.2 Affidabilità . . . 46
5.1.3 Usabilità . . . 46
5.1.4 Efficienza . . . 46
5.1.5 Manutenibilità . . . 46
5.1.6 Portabilità . . . 46
5.2 Suddivisione secondo Standard ISO/IEC 12207:1995 . . . 47
5.2.1 Processi primari . . . 47
5.2.2 Processi di supporto . . . 47
5.2.3 Processi organizzativi . . . 47

6 Metriche di qualità **47**
6.1 Metriche per la qualità di processo . . . 47
6.2 Metriche per la qualità di prodotto . . . 49

1 Introduzione

1.1 Scopo del documento

Lo scopo principale del seguente documento è la definizione delle linee guida che ogni componente del *team*$_G$ deve rispettare per assicurare che il processo di realizzazione del *progetto*$_G$ didattico risulti efficiente ed efficace. Per definire i *processi*$_G$ e le relative *attività*$_G$ del seguente documento si è fatto riferimento allo Standard **ISO/IEC 12207-1995**.

Il documento è organizzato secondo i *processi*$_G$ del ciclo di vita del *software*$_G$ e ogni *processo*$_G$ si configura come una serie di *attività*$_G$, la quale è strutturata attraverso una gerarchia.

E' fondamentale evidenziare che questo documento è in continuo perfezionamento in quanto le norme intrinseche stabilite sono periodicamente rielaborate e ottimizzate.

1.2 Glossario

La documentazione contiene riferimenti al ***Glossario***, all'interno del quale vengono esposti i significati di tutti i termini, che potrebbero risultare troppo specifici o ambigui, pervenuti nei documenti relativi al *progetto*$_G$.
Quando un termine è scritto con una nota a pedice con la lettera *G* indica la possibilità di recuperare la definizione nel ***Glossario***.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- **Standard ISO/IEC 12207-1995** :
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
  Standard$_G$ internazionale che definisce un modello di ciclo di vita del *software*$_G$ e in cui sono definite delle linee guida per la gestione dei *processi*$_G$ *software*$_G$ e le relative *attività*$_G$.

  Stabilisce una struttura per organizzare le diverse fasi e le *attività*$_G$ coinvolte nel ciclo di vita del *software*$_G$, aiutando le organizzazioni a sviluppare prodotti *software*$_G$ in modo più efficiente ed efficace.
  Consultato: 24/02/2025
- **Capitolato C2**: https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C2.pdf
  Consultato: 24/02/2025

1.3.2 Riferimenti informativi

- ***Glossario v2.0.0***;

2 Processi primari

2.1 Fornitura

2.1.1 Introduzione

Il *processo*$_G$ di fornitura di un *software*$_G$ comprende tutte le *attività*$_G$ necessarie per consegnare un prodotto *software*$_G$ funzionante e conforme ai requisiti del *committente*$_G$. Si occupa di controllare e gestire le *attività*$_G$ svolte dal *team*$_G$, per assicurare la copertura di tutti i requisiti e la consegna entro tempi e budget stabiliti.

Il *processo*$_G$ di fornitura viene avviato solo dopo aver completato la stesura del documento di *Valutazione Capitolati*. A seguito viene avviato stipulato un contratto con l'azienda *proponente*$_G$, dove saranno definiti i requisiti e i vincoli per la consegna del prodotto.
Nel resto del capitolo verranno analizzate nel dettaglio le *attività*$_G$ coinvolte, le modalità di comunicazione con l'azienda *proponente*$_G$, la documentazione necessaria e gli strumenti utilizzati per gestire efficacemente il *processo*$_G$ di fornitura del *software*$_G$.

2.1.2 *attività*$_G$

Le *attività*$_G$ coinvolte nella fornitura, secondo quanto stabilito dallo *standard*$_G$ ISO/IEC 12207:1995, del *software*$_G$ sono molteplici e comprendono diverse fasi fondamentali:

1. **Definizione e raccolta dei requisiti**: Identificazione dei bisogni dell'azienda *proponente*$_G$ e definizione dei requisiti funzionali e non funzionali;
2. **Pianificazione del *progetto*$_G$**: Elaborazione di un piano dettagliato con *attività*$_G$, tempistiche e risorse necessarie;
3. **Contrattazione**: Analisi di documenti tecnici, specifiche o capitolati.
4. **Pianificazione**: Vengono definiti i criteri di accettazione, metriche di qualità e strategie di testing.
5. **Esecuzione e controllo**: Progettazione, implementazione e versionamento del codice.;
6. ***test*$_G$ e Verifica**: *test*$_G$ unitari, di integrazione, di *sistema*$_G$ e di accettazione;
7. **Comunicazioni tra *cliente*$_G$ e gli *stakeholder*$_G$**: Riunioni periodiche, aggiornamenti *SAL*$_G$ e risoluzione di problemi;
8. **Deployment e consegna**: Installazione, configurazione del *software*$_G$ nell'ambiente di produzione;

Un'efficace gestione di queste *attività*$_G$ è essenziale per garantire una fornitura del *software*$_G$ di successo, rispettando le tempistiche, il budget e gli *standard*$_G$ di qualità richiesti.

2.1.3 Comunicazioni con l'azienda proponente e documentazione fornita

La comunicazione tra *Byte Your Dreams* e la *proponente*$_G$ **Vimar S.p.A.** sarà un elemento cruciale che permetterà al gruppo di ottenere *feedback*$_G$ su quanto fatto fino a quel momento. Una comunicazione chiara e strutturata consente di ridurre il rischio di malintesi e garantire la correttezza del prodotto. La comunicazione dovrà essere chiara e strutturata ed avverà tramite email per comunicazioni formali o tramite chat di *Microsoft teams* per comunicazioni veloci. Ogni 2 settimane, inoltre, sarà organizzato un incontro in cui si discuterà l'avanzamento del *progetto*$_G$ e gli obbiettivi per il periodo successivo. Ad ogni incontro verrà redatto un verbale esterno, che includerà la data e l'ora dell'incontro, i partecipanti, gli argomenti trattati ed eventuali chiarimenti. I verbali esterni verranno archiviati nella directory "Documenti Esterni/Verbali" all'interno della *repository*$_G$ "Documents" del gruppo Byte Your Dreams.

2.1.4 Documentazione fornita

Di seguito vengono elencati i documenti che verranno forniti ai committenti *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*, e all'azienda *proponente*$_G$ **Vimar S.p.A**.

2.1.4.1 Valutazione dei capitolati

Il documento "Valutazione dei Capitolati" contiene l'analisi dettagliata dei capitolati proposti, con la scelta effettuata dal gruppo *Byte Your Dreams*. Di seguito sono riportate le sezioni del documento:

- **Informativa sulla *proponente*$_G$**: nome azienda, nome *progetto*$_G$, informazioni utili;
- **Obiettivo**: fornisce una sintesi dell'aspettativa del prodotto *software*$_G$ descritto nel capitolato.

- **Tecnologie suggerite**: include suggerimenti e consigli dalla *proponente*$_G$ per lo sviluppo del prodotto.
- **Considerazioni**: include le considerazioni del gruppo *Byte Your Dreams*, i pro, i contro, le criticità e le motivazioni che hanno influenzato la decisione di accettare o meno il capitolato, eventuali proposte di integrazione o modifica.

2.1.4.2 Analisi dei requisiti v2.0.0

L'*analisi dei requisiti*$_G$ fornisce le basi per la progettazione, lo sviluppo e la validazione del *sistema*$_G$. Il documento *Analisi dei Requisiti* fornisce la descrizione dei requisiti del capitolato, dei casi d'uso e delle funzionalità del prodotto finale. Lo scopo di questo documento è definire in modo chiaro e dettagliato i requisiti del capitolato, i casi d'uso e le funzionalità del prodotto. Eliminando così eventuali ambiguità e incomprensioni che potrebbero compromettere la qualità del prodotto finale.
Il documento comprende le seguenti sezioni:

- **Introduzione**: include una breve descrizione del prodotto e delle sue funzionalità.
- **Casi d'uso**: identifica tutti i possibili scenari di utilizzo del prodotto da parte dell'*utente*$_G$. Ogni caso d'uso è accompagnato da una descrizione che ne illustra il funzionamento.
- **Requisiti**: l'insieme di tutte le richieste e i vincoli definiti dalla *proponente*$_G$, o derivanti da discussioni con il gruppo, per la realizzazione del prodotto *software*$_G$G commissionato. Ogni requisito avrà una propria descrizione e il relativo caso d'uso.

2.1.4.3 Piano di progetto v2.0.0

Il documento *Piano di Progetto* descrive in dettaglio come il *progetto*$_G$ sarà realizzato, gestito e controllato, fornendo una roadmap che guida l'intero ciclo di vita del *progetto*$_G$, dalla pianificazione iniziale alla consegna finale del *software*$_G$.
Il documento comprende le seguenti sezioni:

- **Analisi dei rischi**: contiene l'analisi dei rischi che il gruppo prevede di incontrare durante lo svolgimento del *progetto*$_G$. Per la prevenzione dei rischi e/o la loro eventuale mitigazione verrà inserito per ogni possibile rischio una descrizione, una procedura di identificazione, la probabilità di occorrenza, l'indice di gravità e un relativo piano di mitigazione.
- **Pianificazione e metodo utilizzato**: include una serie di accorgimenti per pianificare lo sviluppo delle *attività*$_G$, come: la suddivisione dei ruoli con relativa definizione delle *attività*$_G$ specifiche per ciascun ruolo. Presenta una descrizione dettagliata del modello di sviluppo scelto, insieme alle motivazioni che hanno guidato la decisione verso tale approccio.
- **Preventivo e consuntivo**: contiene una stima delle ore e dei costi previsti per ciascuna *attività*$_G$, suddivisi per ogni periodo stabilito. Alla conclusione di ogni periodo, viene redatto un consuntivo riportante ore e costi sostenuti per ciascuna *attività*$_G$.

2.1.4.4 Piano di qualifica v2.0.0

Il documento *Piano di Qualifica* definisce gli approcci, le metodologie, gli strumenti e le risorse necessari per garantire che il *software*$_G$ sviluppato soddisfi i requisiti di qualità stabiliti, rispettando gli *standard*$_G$ e le aspettative dei *committenti*$_G$ e degli *stakeholder*$_G$.
Il documento contiene le seguenti sezioni:

- **Qualità di processo**: include le strategie di verifica e validazione per garantire che i processi di sviluppo del prodotto raggiungano gli obiettivi di qualità.
- **Qualità di prodotto**: include le strategie di verifica e validazione per assicurare che il prodotto soddisfi gli obiettivi di qualità.
- **Specifica dei test**: include una descrizione dettagliata dei *test*$_G$.

2.1.4.5 Specifica tecnica v1.0.0

Specifica tecnica è un documento che fornisce una descrizione accurata degli aspetti tecnici di **Vimar GENIALE**, concentrandosi sull'architettura logica.
Tale documento specifica le tecnologie utilizzate, le scelte di *design*$_G$ e i *design patterns*$_G$ adottati.
Il suo scopo è quello di fornire una guida chiara e dettagliata per il team di sviluppo, facilitando l'*attività*$_G$ di codifica e di manutenzione del prodotto *software*$_G$.

2.1.4.6 Manuale utente v1.0.0

Il documento *Manuale Utente* funge da guida per gli utenti finali di **Vimar GENIALE**. Fornisce a quest'ultimi le istruzioni necessarie per utilizzare tutte le funzionalità del prodotto *software*$_G$ in modo efficace e ottimale.

2.1.4.7 Glossario v2.0.0

Il *Glossario* offre una definizione chiara e univoca dei termini, concetti e abbreviazioni utilizzati all'interno della documentazione. La presenza di un glossario garantisce ai lettori una comprensione coerente dei concetti specifici presenti. Ogni termine deve essere descritto in modo semplice e preciso, in modo da ridurre eventuali ambiguità e prevenire malintesi.

2.1.4.8 Lettera di presentazione

La *Lettera di Presentazione* accompagna la consegna della documentazione e del prodotto *software*$_G$ durante le fasi di revisione del *progetto*$_G$. Questo documento ha lo scopo di elencare dettagliatamente la documentazione prodotta, che sarà consegnata ai *committenti*$_G$ *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*, e all'azienda *proponente*$_G$ **Vimar S.p.A.**.
Nella *repository*$_G$ "Documents" è possibile consultare le seguenti lettere di presentazione: • Lettera di presentazione Candidatura • Lettera di presentazione RTB

2.1.5 Strumenti

L'adozione di strumenti adeguati è fondamentale per garantire la qualità, l'efficienza e la gestione ottimale di tutte le fasi del *progetto*$_G$. Gli strumenti sono utilizzati in ogni fase del ciclo di vita del *software*$_G$, dalla progettazione alla codifica, dai *test*$_G$ alla gestione del *progetto*$_G$, e contribuiscono a migliorare la produttività, a ridurre il rischio di errori e a facilitare la collaborazione tra i membri del *team*$_G$. Un'accurata selezione degli strumenti giusti è essenziale per affrontare le sfide tecniche e operative che possono sorgere durante lo sviluppo del *software*$_G$. Ecco una panoramica delle principali categorie di strumenti utilizzati durante il *processo*$_G$ di sviluppo *software*$_G$:

- **Microsoft teams**: Un servizio di videoconferenza per gli incontri e una piattaforma di messaggistica istantanea per le comunicazioni con la *proponente*$_{GG}$;
- **Git Flow**: Strategia di branching adottata dal *team*$_G$ che facilita la gestione delle release e lo sviluppo collaborativo tramite l'utilizzo *workflow*$_G$ strutturati
- **GitHub**: *Piattaforme*$_G$ online che offrono *repository*$_G$ *Git*$_G$ collaborative .
- **Visual Studio Code**: Editor di codice che supporta diversi linguaggi di programmazione.

2.2 Sviluppo

2.2.1 Introduzione

Il modello ISO/IEC 12207:1995 fornisce una linea guida per i *processo*$_G$ di sviluppo del *software*$_G$, definendo i processi e le *attività*$_G$ necessarie per produrre *software*$_G$ di alta qualità. La sua adozione consente che ogni fase del processo di sviluppo sia affrontata in modo sistematico, con una chiara separazione delle responsabilità e delle *attività*$_G$, migliorando così la qualità e l'affidabilità del *software*$_G$.

Questo modello si articola in diverse fasi, ognuna con i propri obiettivi e *processi*$_G$. Questo ne garantisce una corretta esecuzione e il rispetto delle normative e degli *standard*$_G$ richiesti.

2.2.2 Analisi dei requisiti

2.2.2.1 Descrizione

L'*analisi dei requisiti*$_G$ è un'*attività*$_G$ fondamentale nello sviluppo di *software*$_G$: costituisce il punto di partenza per il design, l'implementazione e i *test*$_G$ del *sistema*$_G$. Durante questa fase vengono raccolte, documentate, comprese e definite in maniera completa le esigenze dell'*utente*$_G$ e del *sistema*$_G$.
L'obiettivo principale dell'*analisi dei requisiti*$_G$ è quello di assicurarsi che tutte le parti interessate abbiano una visione comune delle funzionalità e delle caratteristiche che il *sistema*$_G$ deve possedere. Per fare ciò è necessario comprendere il dominio, ovvero conoscere i processi, le entità, le regole e le interazioni che caratterizzano quell'area specifica in cui il *sistema*$_G$ sarà applicato. Durante la descrizione dei requisiti, è importante adottare un linguaggio preciso e privo di ambiguità. Ogni requisito deve essere scritto in modo che possa essere facilmente verificato e testato in seguito, attraverso la definizione di criteri di accettazione chiari. È fondamentale che il *processo*$_G$ di descrizione dei requisiti sia iterativo e coinvolga costantemente gli *stakeholder*$_G$. Poiché i requisiti possono evolvere durante il ciclo di vita del *software*$_G$, la revisione e l'aggiornamento del documento dei requisiti è una pratica essenziale per garantire che il prodotto finale soddisfi le reali esigenze degli *utenti*$_G$.

2.2.2.2 Obiettivi

Gli obiettivi dell'*analisi dei requisiti*$_G$ sono strettamente legati alla capacità di tradurre le necessità degli *stakeholder*$_G$ in requisiti chiari, misurabili e realizzabili. In questa fase, si definiscono le linee guida che orienteranno tutte le fasi successive del *progetto*$_G$, dalla progettazione alla codifica. Gli obiettivi devono essere specifici, misurabili, raggiungibili, realistici e tempestivi, in modo che il *team*$_G$ di sviluppo possa monitorare il progresso e garantire che il *software*$_G$ finale risponda efficacemente alle esigenze degli *utenti*$_G$. Gli obiettivi si suddividono in diverse categorie, ognuna delle quali ha un impatto sul successo del *progetto*$_G$.

- Fornire ai progettisti una base chiara e comprensibile per la definizione dell'*architettura*$_G$ e il design del *sistema*$_G$;
- Fornire una base per la pianificazione mediante i requisiti raccolti;
- Facilitare la comunicazione tra fornitori e *proponente*$_G$;
- Fornire riferimenti per la verifica;
- Stabilire priorità;
- Verificabilità: creazione di requisiti verificabili in modo che ogni requisito debba essere definito in modo tale da poter essere testato durante le fasi successive del *progetto*$_G$.

2.2.2.3 Casi d'uso

I casi d'uso rappresentano uno degli strumenti più efficaci per descrivere il comportamento del *sistema*$_G$ in modo chiaro e comprensibile. Forniscono una descrizione dettagliata delle funzionalità del *sistema*$_G$ dal punto di vista degli *utenti*$_G$, identificando, per ogni caso d'uso, una sequenza di azioni che il *sistema*$_G$ esegue in risposta a un'interazione dell'*utente*$_G$ o di un *sistema*$_G$ esterno. La creazione di casi d'uso ha come obiettivo la definizione delle funzionalità del *software*$_G$ dal punto di vista dell'*utente*$_G$ finale, favorendo la comprensione delle specifiche attraverso l'illustrazione chiara e comprensibile di come gli *utenti*$_G$ interagiranno con il *software*$_G$ e quali saranno i risultati di tali interazioni. La gestione e la revisione dei casi d'uso è un'*attività*$_G$ continua. Poiché i requisiti possono evolvere nel tempo, i casi d'uso devono essere aggiornati di conseguenza per riflettere le modifiche nei requisiti e per garantire che il *software*$_G$ finale risponda sempre alle reali necessità degli *utenti*$_G$.
Un caso d'uso include i seguenti elementi:

1. **Identificativo** nel formato: **UC [Numero caso d'uso] . [Numero sotto caso d'uso] - [Titolo]** (es. UC1.2 - Lorem Ipsum). dove:
    - **Numero caso d'uso:** Identificativo del caso d'uso;
    - **Numero sotto caso d'uso:** Identificativo del sottocaso d'uso.
    - **Titolo**: Titolo del caso d'uso.
2. **Attore principale:** Entità che interagisce attivamente con il *sistema*$_G$;
3. **Attore secondario:** Eventuale entità esterna che non interagisce attivamente con il *sistema*$_G$G, ma che permette di soddisfare il bisogno dell'attore principale.
4. **Descrizione:** Descrizione breve della funzionalità;
5. **Scenario principale:** Sequenza di eventi che si verificano quando un attore interagisce con il *sistema*$_G$G per raggiungere l'obiettivo (postcondizioni) del caso d'uso;
6. **Estensioni:** Eventuali scenari alternativi che, nel caso di condizioni diverse, portano il flusso del caso d'uso a non giungere alle postcondizioni;
7. **Precondizioni:** Stato in cui si deve trovare il *sistema*$_G$G affinché la funzionalità sia disponibile all'attore;
8. **Postcondizioni:** Stato in cui si trova il *sistema*$_G$G dopo l'esecuzione dello scenario principale;
9. **User story associata:** Breve descrizione di una funzionalità del *software*$_G$, scritta dal punto di vista dell'*utente*$_G$, che fornisce contesto, obiettivi e valore.

2.2.2.4 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono uno strumento visivo fondamentale per rappresentare in modo grafico le interazioni tra gli attori e il *sistema*$_G$, illustrando i vari flussi di lavoro e i requisiti funzionali. Questi diagrammi sono utilizzati per fornire una visione chiara e intuitiva di come il *sistema*$_G$ dovrebbe comportarsi in risposta agli input degli *utenti*$_G$ o di altri *sistemi*$_G$, offrendo una visione completa delle sue funzionalità. I diagrammi dei casi d'uso sono particolarmente utili per comunicare i requisiti con gli *stakeholder*$_G$ in quanto facilitano la comprensione delle funzionalità.
Riassumendo quindi i casi d'uso:

- Descrivono le funzionalità del *sistema*$_G$ dal punto di vista dell'*utente*$_G$.
- Non forniscono dettagli tecnici sull'implementazione del *sistema*$_G$.
- Mostrano le relazioni e le interazioni tra i diversi casi d'uso.
- Forniscono una panoramica generale del funzionamento del *sistema*$_G$.

I principali componenti di un diagramma dei casi d'uso sono i seguenti:

- **Attori:**
  Gli attori sono rappresentati come figure stilizzate all'esterno del rettangolo che rappresenta il *sistema*$_G$, essi indicano l'entità che interagisce con il *sistema*$_G$. Possono essere persone (utenti finali, amministratori, ecc.), altri *sistemi*$_G$ o dispositivi,

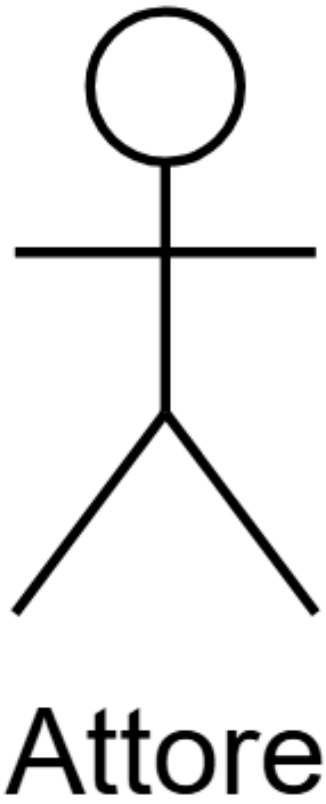

Figure 1: Esempio Attore

- **Casi d'Uso:**
  I casi d'uso sono rappresentati come ovali all'interno del diagramma e indicano le azioni o i comportamenti che il *sistema*$_G$ deve eseguire in risposta agli attori. Ogni caso d'uso descrive una sequenza di azioni che il *sistema*$_G$ segue per soddisfare un obiettivo dell'attore.

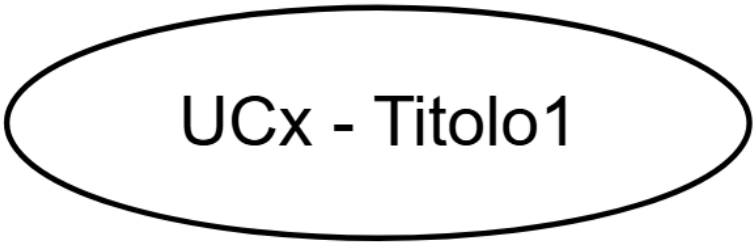

Figure 2: Esempio Caso d'uso

- **Sottocasi d'uso:** I sottocasi d'uso sono dei casi d'uso più specifici e dettaglati, questi offrono maggiori informazioni sulle funzionalità o su particolari scenari rispetto al caso d'uso principale.

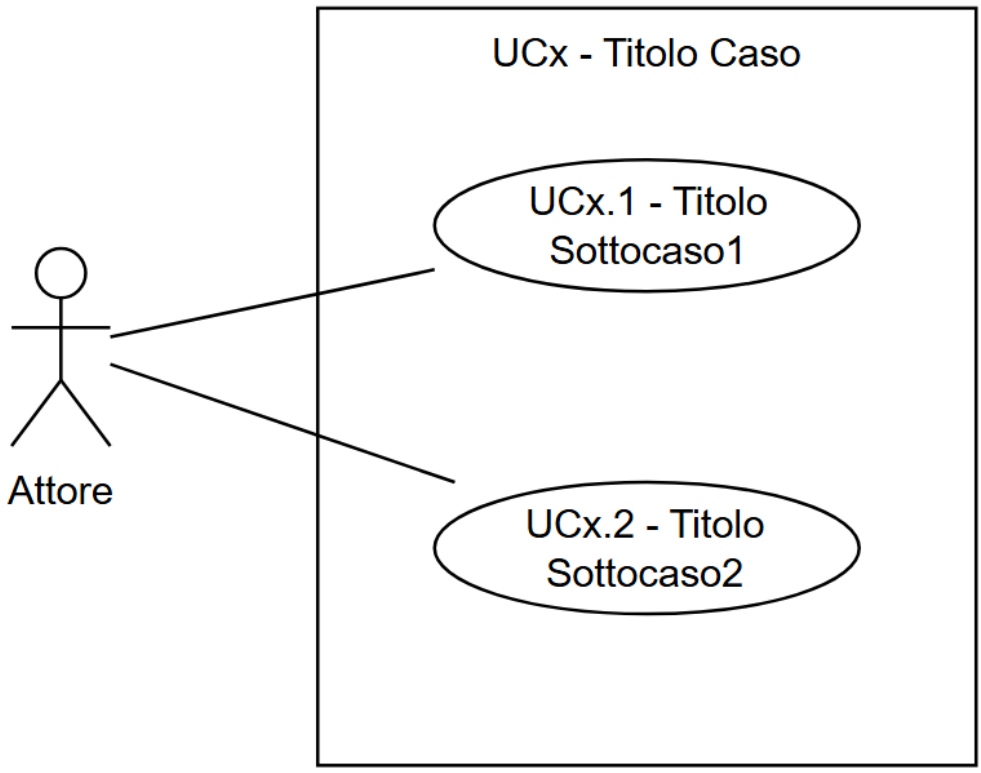

Figure 3: Esempio Sottocaso d'uso

- **Sistema:**
  Il *sistema*$_G$ da modellare è tipicamente rappresentato come un rettangolo che contiene i casi d'uso. Questo confine distingue le azioni e i comportamenti che appartengono al *sistema*$_G$ da quelli esterni ad esso. Gli attori, al di fuori di questo confine, interagiscono con il *sistema*$_G$.

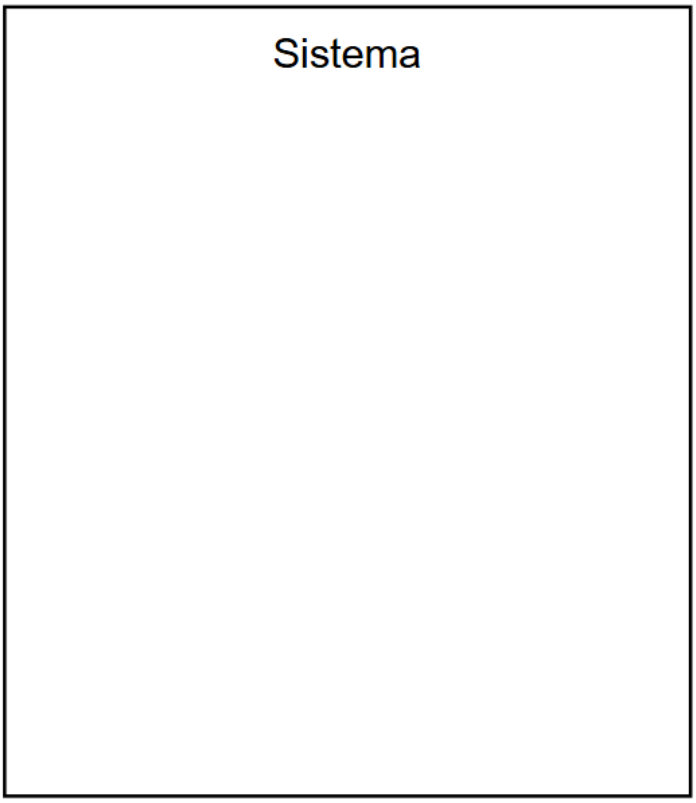

Figure 4: Esempio Sistema

- **Relazioni tra Attori e Casi d'Uso:**
  - **Associazione:**
    Una linea continua collega un attore a un caso d'uso, indicando che l'attore è coinvolto nel caso d'uso.

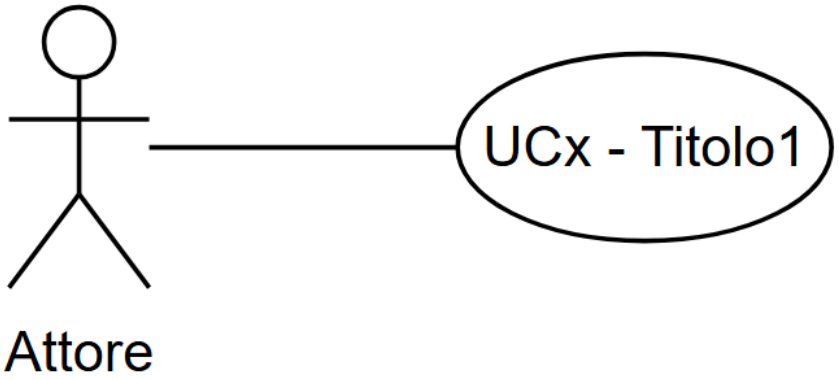

Figure 5: Esempio Associazione

- **Relazioni tra attori:**
  - **Generalizzazione tra attori:** La generalizzazione è usata per mostrare che un attore o un caso d'uso è una specializzazione di un altro, con un comportamento condiviso. Un attore o un caso d'uso derivato eredita i comportamenti e le caratteristiche di quello generale.

Figure 6: Esempio Generalizzazione tra attori

- **Relazioni tra casi d'uso:**
  - **Inclusione:** L'inclusione rappresenta una relazione tra casi d'uso dove un caso d'uso è sempre eseguito come parte di un altro caso d'uso. Questo permette di modularizzare i comportamenti comuni.

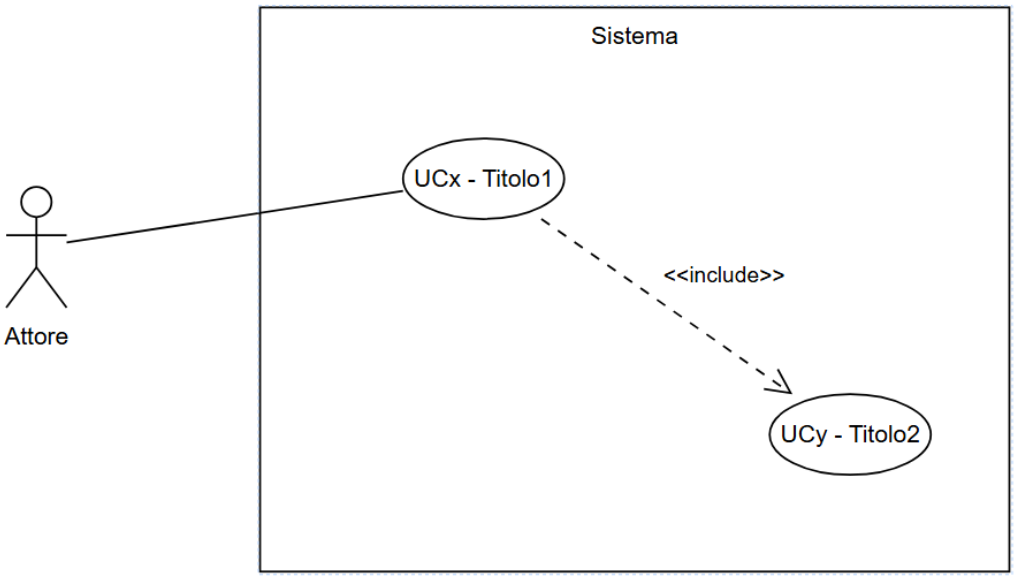

Figure 7: Esempio Inclusione

- **Estensione:** Un'istanza di estensione rappresenta un comportamento opzionale che può essere aggiunto a un caso d'uso in determinate circostanze, a seconda delle condizioni specificate.

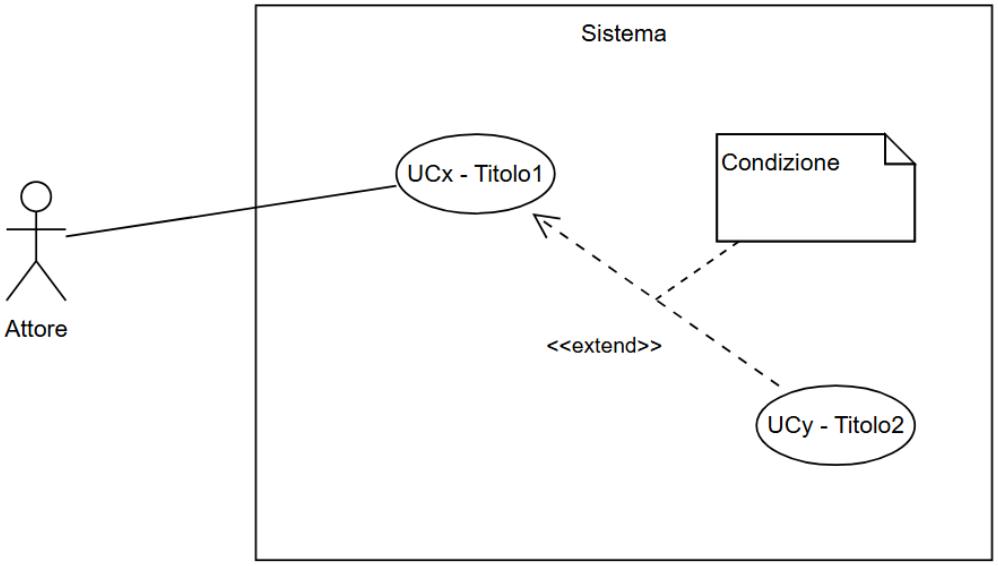

Figure 8: Esempio Estensione

- **Generalizzazione casi d'uso:** Rappresenta una relazione di ereditarietà tra casi d'uso, dove un caso d'uso più specifico eredita il comportamento da uno più generico. È simboleggiata da una linea con una freccia vuota che punta dal caso d'uso più specifico a quello generico.

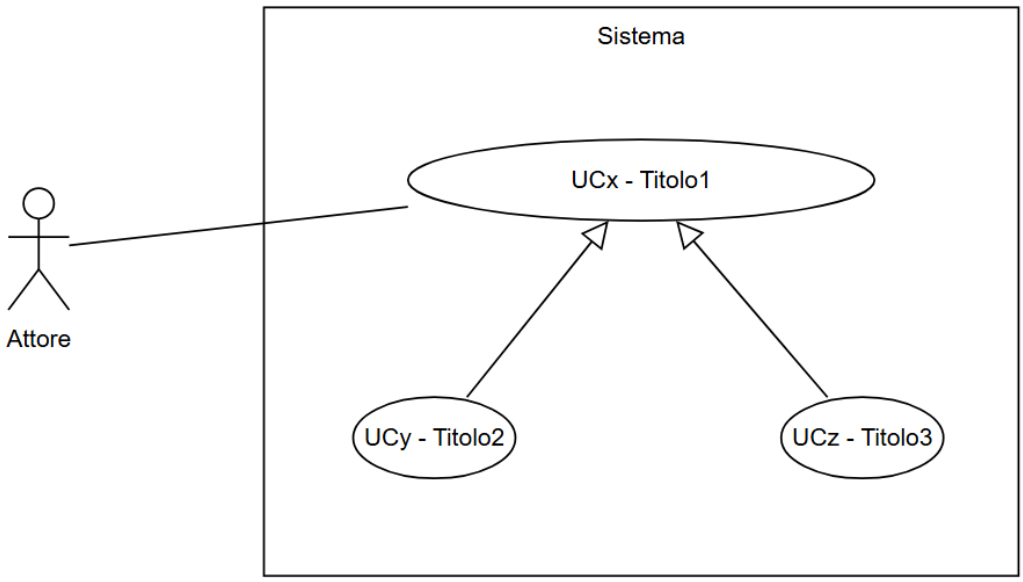

Figure 9: Esempio Generalizzazione

2.2.2.5 Requisiti

I requisiti rappresentano la base su cui si fonda l'intero *processo*$_G$ di progettazione, sviluppo e validazione del *sistema*$_G$. Essi definiscono cosa il *sistema*$_G$ deve fare, come deve comportarsi e le condizioni che devono essere soddisfatte per garantire che il prodotto finale risponda alle necessità degli *utenti*$_G$ e degli *stakeholder*$_G$. I requisiti forniscono una guida chiara e misurabile per il *team*$_G$ di sviluppo e per i tester. Garantiscono che il *sistema*$_G$ risponda alle esigenze degli *utenti*$_G$ e agli obiettivi del *progetto*$_G$. I requisiti possono essere suddivisi in diverse categorie, ognuna delle quali ha un focus specifico:

- **Requisiti funzionali:** I requisiti funzionali descrivono le funzionalità specifiche che il *sistema*$_G$ deve supportare. Essi definiscono il comportamento del *software*$_G$ in termini di azioni, operazioni e risposte a determinate condizioni.
- **Requisiti non funzionali:** Questi requisiti definiscono le caratteristiche qualitative del *sistema*$_G$, come prestazioni, sicurezza, usabilità, compatibilità, scalabilità, manutenibilità e affidabilità. Mentre i requisiti funzionali descrivono "cosa" il *sistema*$_G$ deve fare, i requisiti non funzionali spiegano "come" il *sistema*$_G$ deve comportarsi.

La definizione dei requisiti deve essere chiara ed esaustiva in modo da rispondere pienamente alle aspettative del *committente*$_G$ o del *proponente*$_G$.
Ogni requisito è composto dai seguenti elementi:

1. **Identificativo** nel formato:

   R [Abbreviazione tipologia requisito] [Codice]

   con:

   - **Abbreviazione tipologia requisito:**
     - **RF:** Requisito funzionale;
     - **RQ:** Requisito qualitativo;
     - **RV:** Requisito di vincolo;
   - **Codice:** Numero progressivo all'interno della categoria di requisito.

2. **Importanza:**

- **Obbligatorio:** Essenziale per gli *stakeholder*$_G$;
- **Desiderabile:** Non indispensabile ma auspicabile in quanto aggiungerebbe valore al *sistema*$_G$;
- **Opzionale:** Utile e/o discutibile, da valutare in fase successiva.

3. **Descrizione:** Dettaglio chiaro e completo che illustra il comportamento o la caratteristica che il *software*$_G$ deve possedere;
4. **Fonte:** Fonte del requisito;
5. **Casi d'uso:** Elenco casi d'uso associati.

2.2.2.6 Metriche

Le metriche nell'*analisi dei requisiti*$_G$ permettono di monitorare e misurare vari aspetti del *sistema*$_G$, della sua progettazione e delle sue prestazioni, aiutando i *team*$_G$ di sviluppo a garantire che il *software*$_G$ soddisfi i requisiti stabiliti e che rispetti gli *standard*$_G$ di qualità predefiniti. Le metriche sono utilizzate per analizzare l'efficienza del *processo*$_G$ di sviluppo, la qualità del codice, le prestazioni del *sistema*$_G$ e la soddisfazione degli *utenti*$_G$ finali. Vengono utilizzate per le seguenti motivazioni:

- **Migliorare la qualità**: ci aiutano a prevnire problemi durante lo sviluppo del *sistema*$_G$ trovando errori, omissioni o incongruenze nei requisiti, permettendoci di correggerli.
- **Facilitare la comunicazione**: Le metriche forniscono un linguaggio comune per discutere e valutare i requisiti, semplificando la comunicazione con gli *stakeholder*$_G$.
- **Previsione sviluppo**: Le metriche possono essere utilizzate per prevedere quanto lavoro sarà necessario per implementare i requisiti, aiutandoci a pianificare e gestire il *progetto*$_G$ in modo più efficiente.
- **Monitorare l'avanzamento del *progetto*$_G$**: ci consentono di tracciare quanti requisiti sono stati implementati, aiutandoci a individuare eventuali ritardi o difficoltà.

2.2.3 Progettazione

2.2.3.1 Descrizione

L'*attività*$_G$ di progettazione consente di tradurre i requisiti del *sistema*$_G$, ottenuti nella fase precedente, in un piano d'azione concreto per una soluzione realizzativa ottimale che soddisfi le esigenze degli *stakeholder*$_G$. Questo piano comprende la creazione dell'*architettura*$_G$ del *sistema*$_G$, la definizione delle interazioni tra i suoi componenti, e la progettazione dei dettagli di implementazione che guideranno i programmatori nella fase di codifica. La progettazione è la fase cruciale in cui vengono prese le decisioni più importanti sul comportamento e la struttura del *sistema*$_G$, influenzando direttamente la sua qualità, manutenibilità e performance. Questo permette di:

- perseguire la correttezza per costruzione piuttosto che per correzione
- gestire in modo efficiente la complessità del prodotto per tutta la durata dello sviluppo

2.2.3.2 Obiettivi

Gli obiettivi della fase di progettazione sono essenziali per garantire che il *sistema*$_G$ *software*$_G$ che viene sviluppato soddisfi i requisiti definiti nella fase di analisi e rispetti i vincoli tecnici, economici e di tempo stabiliti. La progettazione è il momento in cui vengono prese decisioni fondamentali che influenzeranno la qualità del prodotto finale, la sua manutenzione e l'evoluzione futura. Gli obiettivi della progettazione vanno oltre la semplice creazione di un *sistema*$_G$ funzionante, cercando di garantire che il *software*$_G$ sia qualitativamente elevato, facilmente manutenibile, scalabile e sicuro, oltre a essere realizzato nel rispetto dei vincoli di tempo e budget. La progettazione è, dunque, un momento chiave per assicurare la solidità del *progetto*$_G$ e il successo a lungo termine del *software*$_G$ sviluppato.
Gli obiettivi principali di questa fase possono essere sintetizzati nei seguenti punti:

- Tradurre i requisiti in soluzioni concrete: L'obiettivo principale della progettazione è quello di tradurre i requisiti funzionali e non funzionali in una struttura ben definita che possa essere realizzata e testata. Questo implica la definizione dell'*architettura*$_G$ del *sistema*$_G$, la suddivisione in moduli e la pianificazione dei flussi di *dati*$_G$ e delle interazioni tra i componenti. Ogni requisito deve trovare una soluzione nel *sistema*$_G$ progettato, garantendo che la qualità e le aspettative degli utenti vengano soddisfatte.
- Definire l'*architettura*$_G$ del *sistema*$_G$: Dopo la selezione delle tecnologie più appropriate, si passa alla progettazione di un'*architettura*$_G$ di alto livello per definire e comprendere la struttura complessiva del prodotto, creando così una base iniziale per lo sviluppo del Proof of Concept (*PoC*$_G$), elemento cruciale della Technology Baseline, per testare le scelte effettuate in merito all'*architettura*$_G$ e alle tecnologie selezionate, oltre a verificarne la conformità con gli obiettivi e le specifiche del *progetto*$_G$. La progettazione ad alto livello si concentra sull'*architettura*$_G$ complessiva del *sistema*$_G$, definendo la struttura dei moduli e come essi interagiranno tra loro. Un'*architettura*$_G$ ben progettata deve essere scalabile, flessibile e in grado di rispondere alle evoluzioni future del *sistema*$_G$. Deve anche tener conto dei vincoli tecnologici, dei tempi di sviluppo e delle risorse disponibili. Infine, si proseguirà con ulteriori iterazioni, effettuando miglioramenti, correzioni e integrazioni, fino a raggiungere un design definitivo. Questo design sarà fondamentale per lo sviluppo dell'MVP (Minimum Viable Product), una versione essenziale e funzionante del prodotto, che farà parte della Product Baseline(*PB*$_G$).
- Ottimizzare la qualità del *software*$_G$: L'*architettura*$_G$ e la progettazione dettagliata devono garantire che il *sistema*$_G$ sia robusto, affidabile e facilmente manutenibile. La qualità del *software*$_G$ non riguarda solo il codice, ma anche la sua capacità di rispondere a variabili come la performance, la sicurezza, la facilità di uso e l'accessibilità. Durante questa fase, si definiscono le modalità di gestione degli errori, dei guasti e delle situazioni critiche.
- Contenere i costi e rispettare i vincoli temporali: La progettazione deve essere anche efficiente dal punto di vista economico, cercando di minimizzare i costi di sviluppo senza compromettere la qualità del *software*$_G$. In particolare, è necessario rispettare i vincoli temporali stabiliti, pianificando

attentamente le fasi del *progetto*$_G$ e cercando soluzioni progettuali che permettano di ridurre il tempo di sviluppo, evitando ritardi o sprechi di risorse.

2.2.3.3 Documentazione

Specifica tecnica
La *specifica tecnica*$_G$ è un documento che descrive in dettaglio il design del *sistema*$_G$ e che fornisce una guida chiara e dettagliata per gli sviluppatori per la corretta implementazione della soluzione secondo i requisiti e le specifiche indicate. Questo documento è fondamentale in quanto riduce la complessità e l'ambiguità dello sviluppo del prodotto, e garantisce l'allineamento di quest'ultimo alle aspettative del cliente.
All'interno si possono trovare informazioni relative a:

- **Tecnologie utilizzate**: specifica tecnologie, librerie e framework integrati nel *sistema*$_G$ e le motivazioni per cui sono stati scelti;
- **Architettura logica**: descrive i componenti, i ruoli, le connessioni e le interazioni tra i vari moduli del *sistema*$_G$;
- **Architettura di deployment**: descrive l'architettura fisica del *sistema*$_G$ e le modalità di distribuzione dei vari componenti;
- **Pattern architetturali e design pattern**: definisce i pattern architetturali e i design pattern utilizzati e quelli influenzati dalle tecnologie scelte;
- **Requisiti tecnici**: definisce i requisiti tecnici, di scalabilità e di conpatibilità che il prodotto deve soddisfare.

2.2.3.4 Qualità dell'architettura

- **Modularità:** L'*architettura*$_G$ deve essere suddivisa in moduli o componenti ben definiti, ognuno con una responsabilità chiara. Questo facilita la comprensione, la manutenzione e la riutilizzabilità del codice.
- **Scalabilità:** L'*architettura*$_G$ deve essere progettata in modo da poter gestire un aumento del carico di lavoro o dell'utenza senza compromettere le prestazioni. Ciò può includere la possibilità di aggiungere nuovi moduli o componenti senza dover riprogettare l'intero *sistema*$_G$.
- **Flessibilità:** L'*architettura*$_G$ deve essere in grado di adattarsi a cambiamenti nei requisiti o nelle tecnologie senza richiedere una revisione completa. Ciò può includere l'uso di interfacce ben definite e la separazione delle preoccupazioni.
- **Sicurezza:** L'*architettura*$_G$ deve essere progettata tenendo conto della sicurezza, proteggendo i dati sensibili e garantendo che le vulnerabilità siano minimizzate. Ciò può includere l'uso di crittografia, autenticazione e autorizzazione adeguate.
- **Affidabilità:** L'*architettura*$_G$ deve essere progettata per garantire che il *sistema*$_G$ funzioni correttamente anche in caso di errori o guasti. Ciò può includere l'uso di meccanismi di tolleranza agli errori e la gestione delle eccezioni.
- **Disponibilità:** L'*architettura*$_G$ deve garantire che il *sistema*$_G$ sia disponibile e accessibile agli *utenti*$_G$ quando necessario. Ciò può includere l'uso di bilanciamento del carico, ridondanza e failover.
- **Riusabilità:** L'*architettura*$_G$ deve essere progettata in modo da consentire il riutilizzo di componenti o moduli in altri *sistemi*$_G$. Ciò può includere l'uso di librerie e framework ben definiti.
- **Semplicità:** L'*architettura*$_G$ deve essere progettata in modo da essere il più semplice possibile, evitando complessità inutili. Ciò facilita la comprensione e la manutenzione del *sistema*$_G$.
- **Incapsulazione:** L'*architettura*$_G$ deve garantire che i dettagli interni di un modulo siano nascosti agli altri moduli, consentendo solo l'accesso alle interfacce pubbliche. Riducendo così le dipendenze tra i moduli e facilitando la manutenzione.

- **Coesione:** la misura in cui i componenti all'interno di un modulo sono strettamente correlati tra loro. Un'alta coesione indica che un modulo ha una singola responsabilità e che le sue funzionalità sono correlate, facilitando la comprensione e la manutenzione del codice.
- **Basso accoppiamento:** la misura in cui i moduli sono indipendenti l'uno dall'altro. Un basso accoppiamento indica che i moduli possono essere modificati o sostituiti senza influenzare gli altri moduli, facilitando la manutenzione e l'evoluzione del *sistema*$_G$.

2.2.3.5 Diagrammi UML

Strumenti che rappresentano graficamente la struttura e il comportamento del *sistema*$_G$. Tali diagrammi offrono i vantaggi:

- **Standardizzazione**: I diagrammi UML seguono uno standard riconosciuto a livello internazionale, il che significa che possono essere compresi da sviluppatori e progettisti in tutto il mondo;
- **Facilitare la comunicazione**: I diagrammi UML forniscono un linguaggio visivo comune che facilita la comunicazione tra i membri del *team*$_G$ e gli *stakeholder*$_G$;
- **Analisi e progettazione**: I diagrammi UML aiutano a visualizzare e analizzare i requisiti del *sistema*$_G$, eventuali errori e/o lacune del progetto prima dell'implementazione finale;
- **Facilitare la manutenzione**: I diagrammi UML possono essere utilizzati come documentazione di riferimento per la manutenzione e l'evoluzione del *sistema*$_G$ nel tempo;
- **Supporto alla documentazione**: I diagrammi UML vengono utilizzati all'interno della documentazione per fornire una guida visiva dell'intero *sistema*$_G$.

Per la progettazione del *sistema*$_G$ sono stati utilizzati i **diagrammi delle classi**.

Diagrammi delle classi

Ogni diagramma delle classi rappresenta le proprietà, le operazioni e le relazioni tra le varie componenti di un *sistema*$_G$.
Le classi sono rappresentate come rettangoli divisi in tre sezioni:

1. **Nome della classe:** La parte superiore del rettangolo contiene il nome della classe, che deve essere significativo e rappresentativo del suo scopo.
2. **Attributi:** La sezione centrale elenca gli attributi o le proprietà della classe, con i loro tipi e visibilità (pubblica [+], privata [-], protetta [#]).
3. **Metodi:** La parte inferiore contiene i metodi o le operazioni che la classe può eseguire, con i loro parametri e tipi di ritorno.

Convenzioni sui metodi

- I metodi astratti sono scritti in corsivo;
- I metodi getter, setter e i costruttori non sono inclusi fra i metodi;

Relazioni tra classi

Le relazioni tra le classi sono rappresentate da linee che collegano i rettangoli delle classi. Le principali relazioni includono:

- **Associazione:** Rappresenta una relazione tra due classi, indicata da una linea continua. Può essere unidirezionale o bidirezionale.
- **Dipendenza:** Indica che una classe dipende da un'altra per il suo funzionamento. È rappresentata da una linea tratteggiata con una freccia.

- **Aggregazione:** Indica una relazione "part of", in cui una classe contiene un'altra classe. È rappresentata da una linea con un rombo vuoto all'estremità del tutto.
- **Composizione:** È una forma più forte di aggregazione, in cui la parte non può esistere senza il tutto. È rappresentata da una linea con un rombo pieno all'est
- **Generalizzazione:** Indica una relazione di ereditarietà tra classi, in cui una classe figlia eredita le proprietà e i metodi di una classe padre. È rappresentata da una linea con una freccia vuota.
- **Interface Realization:** Indica che una classe implementa un'interfaccia. È rappresentata da una linea tratteggiata con una freccia vuota.

2.2.3.6 Design pattern

I design pattern sono soluzioni generiche e riutilizzabili a problemi comuni che si presentano durante lo sviluppo del *sistema*$_G$. Questi pattern forniscono linee guida e best practices per affrontare situazioni specifiche, migliorando la qualità del codice e facilitando la manutenzione.
I design pattern possono essere classificati in tre categorie principali:

- **Creazionali:** Si occupano della creazione di oggetti e delle modalità di istanziazione. Esempi includono il Singleton, il Factory Method e l'Abstract Factory.
- **Strutturali:** Si concentrano sulla composizione di classi e oggetti per formare strutture più grandi. Esempi includono l'Adapter, il Composite e il Decorator.
- **Comportamentali:** Si occupano della comunicazione tra oggetti e della gestione del comportamento del sistema. Esempi includono il Observer, il Strategy e il Command.

Questa documentazione favorisce una comprensione dell'integrazione del *design pattern*$_G$ nell'*architettura*$_G$ generale.

2.2.3.7 Strumenti

- **StarUML:** applicazione che permette di creare diagrammi UML in modo semplice e intuitivo. Supporta vari tipi di diagrammi, tra cui diagrammi delle classi, diagrammi di sequenza e diagrammi di attività. StarUML offre anche funzionalità avanzate come la generazione automatica di codice e la personalizzazione dei diagrammi.

2.2.4 Codifica

2.2.4.1 Descrizione e Scopo

La codifica è la fase in cui la progettazione del *sistema*$_G$, precedentemente definita, prende vita attraverso la scrittura effettiva del codice *sorgente*$_G$. In questa fase, gli sviluppatori traducono le specifiche progettuali in linguaggi di programmazione, creando il *software*$_G$ che soddisferà i requisiti funzionali e non funzionali del *sistema*$_G$. Questa fase richiede precisione, attenzione ai dettagli e aderenza alle linee guida stabilite durante la fase di progettazione. È anche una fase in cui gli sviluppatori devono considerare non solo la corretta implementazione delle funzionalità, ma anche l'efficienza, la sicurezza e la manutenibilità del codice. Verranno applicate le best practices per la creazione del codice, garantendone la mantenibilità, la leggibilità e la comprensibilità secondo il "Clean Code". Tutto queso verrà fatto uniformandosi alle metriche definite nel Piano di Qualifica.

2.2.4.2 Aspettative

Ci si aspetta un *software*$_G$ che sia in grado di soddisfare le pretese e le esigenze della *proponente*$_G$, il codice dovrà essere fedele alle specifiche e dovrà avere le prestazioni ottimizzate. Il codice dovrà anche essere accompagnato dai relativi *test*$_G$ che ne dovranno garantire il corretto funzionamento e l'esattezza.

2.2.4.3 Norme di codifica

Le norme di codifica sono un insieme di linee guida e best practices che gli sviluppatori devono seguire durante la scrittura del codice. Queste norme furono enunciate da Robert C. Martin nel libro "Clean Code". Esse assicurano che il codice sia leggibile, mantenibile, e facilmente comprensibile da altri sviluppatori, anche in futuro. Le norme utilizzate dal gruppo sono le seguenti:

- **Nomi significativi:** I nomi delle variabili, delle classi e delle funzioni devono essere significativi e autoesplicativi, in modo da rendere chiaro il loro scopo e il loro utilizzo.
- **Indentazione:** Utilizzare uno stile di indentazione uniforme per facilitare la lettura e la comprensione del codice.
- **Lunghezza delle funzioni:** Le funzioni devono essere brevi e focalizzate su un'unica responsabilità, in modo da rendere il codice più leggibile e manutenibile.
- **Commenti:** Utilizzare commenti per spiegare il codice complesso o per fornire informazioni aggiuntive, evitando commenti ridondanti o superflui.

2.2.5 Configurazione dell'ambiente di esecuzione

2.2.5.1 Docker

I container offrono un ambiente isolato e leggero che consente di eseguire *applicazioni*$_G$ in modo coerente, indipendentemente dall'ambiente sottostante. Le regole utilizzate dai programmatori per la scrittura dei file *Docker*$_G$ sono le seguenti:

- **Scrivere file ottimizzati:** minimizzare il numero di istruzioni per minimizzare i layer dell'immagine;
- **Utilizzare nomi descrittivi per i *container*$_G$**: i nomi dei *container*$_G$ devono riflettere in modo chiaro il loro contenuto o la loro funzione;
- **Utilizzare versioni specifiche:** specificare sempre la versione dell'immagine di base utilizzata nel Dockerfile, evitando l'uso di "latest";
- **Utilizzare immagini ufficiali:** utilizzare immagini ufficiali o verificate da fonti affidabili;
- **Evitare *processi*$_G$ con privilegi elevati**;
- **Utilizzo variabili ARG**: Utilizzare variabili ARG per gestire informazioni sensibili;
- **Ottimizzare le risorse**: Impostare dei limiti di utilizzo di risorse.
- **Utilizzare variabili d'ambiente**: Utilizzare variabili d'ambiente per configurazioni dinamiche, fornendo valori predefiniti;
- **Configurazione loggin**: Configurare i *container*$_G$ per registrare efficacemente i *log*, integrando strumenti di monitoraggio se necessario;
- **Integrazione Test**: Integrare i *test*$_G$ nell'immagine del *container*$_G$;

2.2.5.2 Strumenti

Gli strumenti principali utilizzati nel processo includono:

- **Visual Studio Code**: Editor di codice che supporta diversi linguaggi di programmazione.
- **GitHub Copilot**: Strumento che usa l'*AI*$_G$ che assiste nella scrittura del codice.
- **Docker**: *Piattaforma*$_G$ per la *containerizzazione*$_G$ attraverso le immagini docker, usata per garantire la portabilità e la consistenza degli ambienti di sviluppo e produzione.

3 Processi di supporto

3.1 Documentazione

3.1.1 Introduzione

La documentazione è l'informazione che accompagna un prodotto *software*$_G$. Descrivere il prodotto a sviluppatori, distributori e *utenti*$_G$, in modo da facilitare l'*attività*$_G$ di sviluppo e da permettere una comprensione del prodotto stesso senza un supporto umano.
Grazie alla documentazione, vengono dettate regole precise e dirette che assicurano che i *processi*$_G$ si svolgano con la qualità attesa, e la semplificazione della manutenzione del prodotto.

Questa sezione definisce delle procedure ripetibili che hanno l'obbiettivo di uniformare la documentazione e di semplificare la redazione dei documenti. Tali procedure dovranno essere applicate da tutti i membri del *team*$_G$. I file sorgenti LaTeX con tale documentazione saranno inseriti nella *repository*$_G$ privata *DocsSource*.

3.1.2 Ciclo di vita

Ogni documento segue le seguenti fasi:

1. Si crea un *branch*$_G$ per lo sviluppo del documento/sezione nella *repository*$_G$ *DocsSource* attraverso i seguenti comandi:

   ```
   git checkout develop
   git flow feature start id_FEATURE
   ```

2. Si copia all'interno della cartella appropriata il *template* del documento/sezione da redigere;
3. Si redige il/la documento/sezione assegnata.
4. Si aggiorna il registro delle modifiche, incrementando la versione e aggiungendo i dati in una nuova riga. Quest'azione deve seguire la convenzione descritta nella sezione X.Y.Z e tramite "X.Y.Z".
5. Si rende accessibile il *branch*$_G$ nella *repository*$_G$ attraverso i comandi:

   ```
   git add .
   git commit -m "Descrizione commit"
   git push origin feature/Id_FEATURE
   ```

6. Si segnala il completamento dell'*attività*$_G$ spostando l'issue nella colonna "In review"
7. Si crea una Pull Request, selezionando come *branch*$_G$ di destinazione quella "main" e come *branch*$_G$ di origine quello dedicato alla redazione del documento, o della sezione.;
8. Si clicca su "Create Pull Request"
9. Si inserisce un titolo significativo e una breve descrizione, si selezionano i verificatori su "Reviewers" e infine si clicca su "Create Pull Request".
10. Per modifiche richieste dal verificatore, o dai verificatori, si ripetono i punti, in ordine, dal punto 3 al punto 5;
11. Si elimina il *branch*$_G$ quando la pull request viene chiusa o risolta.

3.1.3 Sorgente documenti

Per consentire a più Redattori di lavorare su un singolo file, è stato decidere di creare singoli file per ogni sezione/sottosezione individuata dal Responsabile. Così facendo, si evita la modifica di parti del documento che non sono assegnate al singolo Redattore.
Tutti questi file vengono collegati in un file principale col nome del documento. Il collegamento verrà effettuato mediante il comando `\input{Sezione.tex}` o `\input{Sottosezione.tex}`.
Nel caso dei verbali non si usa questo approccio date le sezioni brevi.

3.1.4 Nomenclatura dei documenti

Il nome dei documenti deve essere omogeneo: la prima lettera deve essere minuscola, mentre le restanti minuscole. Tranne per i verbali, nel nome del documento deve essere riportata anche la versione attuale del documento (vedi convenzione per il versionamento, sezione X.Y.Z).
Nello specifico la convenzione da seguire è la seguente:

- **Verbali:**
  - **Interni:** VerbaleInterno_AAAAMMDD;
  - **Esterni:** VerbaleEsterno_AAAAMMDD;
- **Norme di Progetto**: Norme_di_progetto_vX.Y.Z;
- **Analisi dei requisiti**: Analisi_dei_requisiti_vX.Y.Z;
- **Piano di progetto**: Piano_di_progetto_vX.Y.Z;
- **Glossario**: Glossario_vX.Y.Z .

3.1.5 Struttura del documento

I documenti ufficiali seguono una struttura precisa e comune, rigorosamente rispettata.

3.1.5.1 Prima pagina

Nella prima pagina è presente:

- Logo, nome e mail del gruppo;
- Nome del documento;
- Redattori;
- Verificatori;
- Amministratore;
- Destinatari;
- Firma digitale del responsabile.

3.1.5.2 Registro delle modifiche

Ogni documento deve essere provvisto di un registro delle modifiche tabellare che contiene un riassunto delle modifiche apportate al documento nel corso del tempo.
La tabella viene inserita nella sezione *registro delle modifiche*, prima dell'*indice*. All'interno vengono riportate le seguenti informazioni:

- Versione del file;
- Data di rilascio;
- Autore;
- Verificatore;
- Descrizione: sintesi delle modifiche apportate.

Informazioni utili:

- La prima riga della tabella identifica la versione attuale del documento. Per permetterne l'integrazione automatica nel nome del Documento, durante la compilazione del file *.tex*, è necessario utilizzare il comando:
  `{\label{Git_Action_Version} X.Y.Z.}`.
- La convenzione per il versionamento è presente alla sezione 3.*.*

3.1.5.3 Indice

Ogni documento deve contenere un indice, posto nella pagina seguente al *Registro delle modifiche*. All'interno saranno elencate le sezioni e le sottosezioni, ognuna delle quali raggiungibile tramite click.

3.1.5.4 Piè di pagina

In ogni piè di pagina deve essere presente il logo del gruppo e il numero di pagina.

3.1.5.5 Verbali

Questi documenti vogliono riportare gli oggetti di discussione, le decisioni adottate, le azioni da intraprendere e le figure coinvolte, durante i meeting.
Si distinguono in verbali interni ed esterni. I primi sono destinati all'uso interno dell'organizzazione, mentre i secondi trovano applicaione quando nelle discussioni o nelle decisioni vengono coinvolte terze parti.
La struttura dei verbali è la seguente:

- **Prima pagina** Oltre alle informazioni comuni a ogni documento vengono riportate:
  - Data e tipologia della riunione nel nome del documento
  - Luogo fisico o Canale di comunicazione adottato
  - Ora di inizio e fine
  - Partecipanti della riunione (interni ed in caso esterni)
  - Il Responsabile
- **Corpo del documento** Sono presenti le sezioni:
  - **Revisione del periodo precedente** Viene analizzato lo stato delle *attività*$_G$, aspetti positivi ed eventuali difficoltà incontrate, in modo da identificare azioni migliorative per ottimizzare i *processi*$_G$.
  - **Ordine del giorno** Vengono elencate le tematiche discusse durante la riunione, con le relative decisioni.
    Se sono presenti richieste di chiarimenti efftuate verso le terze parti coinvolte, vengono incluse nella sezione "**Richieste e chiarimenti**".
  - **Attività da svolgere** Tabella in cui vengono identificate le *attività*$_G$ e in cui si specifica:
    - Nome della task
    - Id *issue*$_G$
    - Verificatore a cui è destinata l'*attività*$_G$

3.1.6 Regole stilistiche

Regole sintattiche:

- I titoli delle sezioni iniziano con la lettera maiuscola;
- Le date vengono scritte nel formato GG/MM/AAAA (giorno/mese/anno);
- Ogni voce dell'elenco deve terminare con un punto e virgola(**;**), tranne l'ultima che, invece, deve terminare con un punto(**.**);

- I numeri razionali devono essere scritti con la virgola;
- Quando si fa riferimento a informazioni contenuti in documenti diversi, si devono indicare questi ultimi, con la relativa versione e sezione.(Ex: Norme di Progetto v2.0.0 sez. 1.2.3).

Stile del testo:

- **Grassetto**:
  - Titoli delle sezioni/sottosezioni/paragrafi;
  - Parole/frasi considerate di rilievo;
  - Nomi di Aziende.
- **Italico**:
  - Riferimenti a paragrafi, sezioni e documenti;
  - Termini presenti nel Glossario;
  - Nomi delle aziende.

Riferimenti:

- Per quelli **interni** si usa il colore RGB(R,G,B);
- Per quelli **esterni** si usa il colore: RGB(R,G,B);

3.1.7 Strumenti

Gli strumenti usati per la redazione dei documenti sono i seguenti:

- **GitHub:** serve per il versionamento, la collaborazione, la distribuzione e l'automatizzazione della compilazione;
- **LaTeX:** serve per la redazione dei documenti;
- **Visual Studio Code:** IDE utilizzato per la redazione dei documenti. Viene utilizzato il *plugin*$_G$ **LaTeX Workshop**

3.2 Verifica

3.2.1 Scopo

La verifica è un processo che guida l'intero ciclo di vita *software*$_G$, e che garantisce efficienza e correttezza nelle *attività*$_G$. Il suo scopo è quello di assicurare che i prodotti siano conformi rispetto ai requisiti. Durante

la fase di Verifica viene analizzata la qualità del prodotto e dei *processi*$_G$. Affinché il prodotto passi alla fase successiva (quella di Validazione) è necessario seguire convenzioni chiare e dettagliate, in modo tale da garantire le qualità attese dettate da *standard*$_G$ definiti nel documento *Piano di Qualifica*.

Per permettere ai Verificatori di valutare la qualità del prodotto e attuare ogni *attività*$_G$ di verifica, ognuna di queste viene documentata all'interno del documento *Piano di Qualifica*.

3.2.2 Verifica dei documenti

Il Verificatore quando visualizza un'*issue*$_G$ nella colonna "In Review" della *dashboard*$_G$ (da mettere link), deve convalidare il file corrispondente nella *repository*$_G$ privata *DocsSource*, garantendo la qualità e l'accuratezza del contenuto.

Dopo che il redattore avrà creato la Pull Request, il revisore troverà la richiesta di unire il *branch*$_G$ dedicato alla redazione del documento o di una sua sezione, al *branch*$_G$ "develop".

Durante questa fase il revisore dovrà esaminare ciò che è stato fatto, e in caso di necessità, potrà fornire *feedback*$_G$ e suggerimenti ai redattori per eventuali modifiche.

Affinché un documento passi alla fase di valutazione, devono essere fatte le seguenti verifiche:

- **Revisione tecnica**: le informazioni all'interno del documento devono essere corrette, coerenti e devono rispettare le norme stabilite;
- **Conformità alle norme**: il documento deve seguire gli *standard*$_G$ prestabiliti per lo stile, la formattazione e la struttura;
- **Consistenza e coerenza**: il documento non deve presentare discrepanze o contraddizioni rispetto a quanto scritto all'interno, e a quanto scritto in altri documenti;
- **Revisione grammaticale e ortografica**: il documento non deve presentare errori grammaticali, ortografici e di punteggiatura;
- **Chiarezza e comprensibilità**: il contenuto e il linguaggio usato devono essere chiari e comprensibili ai destinatari.

Procedimento di convalidazione

1. **Accettazione della Pull Request**: si accede alla pagina della Pull Request, si risolvono eventuali conflitti, e infine si effettua il Merge Pull Request;
2. **Eliminanazione del branch**: si elimina il *branch*$_G$ di feature creato per la redazione del documento o della sezione;
3. **Spostamento della issue in Done**: nella *dashboard*$_G$(link) si sposta la relativa issue dalla colonna "In Review" a quella "Done";

Solamente quando il documento avrà raggiunto un certo livello di maturità, verrà aperta una nuova Pull Request destinata al responsabile per la sua convalida e per il suo merge all'interno del branch "main". Quest'ultimo passaggio attiverà la *GitHub Action*$_G$ che andrà a generare automaticamente il PDF del documento, con la relativa versione, all'interno della repository "Documents". Nel caso in cui la compilazione dovesse fallire, il responsabile dovrà accedere alla sezione "Actions" di GitHub, esaminare l'origine dell'errore e correggerlo.

3.2.3 Analisi

3.2.3.1 Analisi statica

Tipologia di analisi che mira a individuare problemi o irregolarità all'interno del prodotto senza la sua esecuzione. Gli errori che intende individuare sono di tipo formale, come la presenza di difetti/proprietà indesiderati/e, o come la violazione di vincoli imposti.
Le tecniche impiegate dal gruppo sono:

Walkthrough
Metodologia che analizza sistematicamente la documentazione e/o il codice, e che promuove la collaborazione tra Autore e Verificatore. Questo approccio è preferito durante le fasi iniziali del progetto, dove si trovano prodotti poco complessi.

Inspection
Tecnica strutturata composta da una sequenza di passaggi ben definiti, che mira a rilevare difetti specifici all'interno di un documento o del codice.
Gli elementi da verificare sono organizzati in liste di controllo, ognuna documentata all'interno del documento *Piano di qualifica*.

In fasi avanzate del progetto, con la presenza di prodotti complessi, la verifica di quest'ultimi risulta più efficace tramite inspection rispetto a walkthrough.

3.2.3.2 Analisi dinamica

Tipologia di analisi applicabile al solo *software*$_G$ che verifica il corretto funzionamento del codice, attraverso la sua esecuzione. Durante tale processo vengono eseguiti una serie di *test*$_G$, derivati dai requisiti (funzionali e non), che assicurano l'esecuzione accurata del *software*$_G$ e che garantiscono risultati ottenuti corrispondenti a quelli attesi.

Per garantire efficacia, questo processo deve essere automatizzato e ripetibile, in modo da permettere una valutazione oggettiva del prodotto. La definizione e l'esecuzione dei *test*$_G$ seguono i principi del *Modello a V*$_G$.

3.2.3.3 Test di unità

Test$_G$ progettati per la verifica delle singole componenti/unità, del *sistema*$_G$ che ne garantiscono il corretto funzionamento.
Si suddividono in 2categorie:

- **Test funzionali**: verificano che ogni unità esegua le funzioni specificate nell'implementazione e che produca i risultati desiderati.
- **Test Strutturali**: verificano la struttura interna, la logica di controllo e il flusso dei dati dell'unità attraverso l'esaminazione del codice.

3.2.3.4 Test di integrazione

Test$_G$, pianificati durante la fase di progettazione architetturale, cruciali nell'analisi dinamica del *software*$_G$ che valutano il corretto funzionamento delle unità quando combinate. Più precisamente verificano se quest'ultime collaborano in maniera efficacie.
L'obbiettivo di tali *test*$_G$ è di assicurare che il software funzioni perfettamente e secondo le specifiche del progetto.

3.2.3.5 Test di Sistema

Test$_G$, definiti durante l'analisi dei requisiti, che verificano la copertura dei requisiti, valutando interamente il *software*$_G$ e accertando le corrette integrazioni tra le unità.

Vengono progettati durante l'analisi dei requisiti, ed eseguiti solo dopo il completamento dei *test*$_G$ di *integrazione*$_G$.

3.2.3.6 Test di accettazione

Test$_G$ fondamentali per il rilascio del *software*$_G$. Garantiscono che quest'ultimo soddisfi le aspettative, e che risponda ai requisiti specificati.

3.2.3.7 Sequenza delle fasi dei test

La sequenza delle fasi di *test*$_G$ è così composta:

1. Test di Unità;
2. Test di Integrazione;
3. Test di Sistema;
4. Test di Accettazione.

3.2.4 Classificazione dei test

Ogni *test*$_G$ è indentificato in base alla propria tipologia e atttraverso un codice identificativo nel seguente formato:

T [Tipologia Test] [Codice]

dove:

- **[Tipologia Test]:**
  - U: *test*$_G$ di Unità;
  - I: *test*$_G$ di Integrazione;
  - S: *test*$_G$ di Sistema;
  - A: *test*$_G$ di Accettazione.
- **[Codice]**: numero progressivo, nel caso il test non avesse un padre; mentre nel caso in cui il *test*$_G$ abbia un padre, avrà il seguente formato:

  [Codice.padre].[Codice.figlio]

  dove il **[Codice.padre]** identifica univocamente il padre del *test*$_G$; mentre **[Codice.figlio]** è il numero progressivo che identifica il *test*$_G$.

3.2.5 Stato dei test

Nella sezione relativa ai test nel documento *Piano di Qualifica* per ogni test viene registrato il suo risultato.
Lo stato di un *test*$_G$ può essere:

- **N-I**: il *test*$_G$ non è stato implementato;
- **S**: il *test*$_G$ ha dato esito positivo;
- **N-S**: *test*$_G$ ha dato esito negativo.

3.2.6 Strumenti

- **Modulo "unittest" Python**: liberia *standard*$_G$ che permette di creare, organizzare ed eseguire *test*$_G$ di unità efficacemente e in modo automatizzato.

3.3 Validazione

3.3.1 Introduzione

Nel contesto dell'ingegneria del *software*$_G$, la validazione rappresenta una fase cruciale per verificare che il prodotto sviluppato soddisfi pienamente le aspettative di chi lo ha commissionato. Questo passaggio è fondamentale per accertarsi che il *software*$_G$ risponda agli obiettivi e ai requisiti definiti nelle fasi iniziali del *progetto*$_G$.
Un aspetto essenziale della validazione è il confronto diretto con il *committente*$_G$ e gli *stakeholder*$_G$, utile per raccogliere *feedback*$_G$ e assicurare che quanto realizzato sia in linea con le aspettative degli utenti finali.
Il prodotto conclusivo deve quindi:

- Soddisfare tutti i requisiti concordati con il *committente*$_G$;
- Funzionare correttamente nell'ambiente operativo previsto.

Lo scopo finale è garantire che il prodotto sia pronto per il rilascio, segnando così il completamento del ciclo di vita del *progetto*$_G$.

3.3.2 Procedura di validazione

La procedura di validazione si basa sui *test*$_G$ pianificati durante l'*attività*$_G$ di verifica, come descritti nelle *Norme di Progetto*. L'elemento centrale di questa fase è rappresentato dal *test di accettazione*$_G$, che serve a confermare la conformità del prodotto rispetto ai requisiti.
I test devono garantire:

- La copertura dei casi d'uso previsti;
- Il soddisfacimento dei requisiti obbligatori;
- La conformità con gli ulteriori requisiti concordati con il *committente*$_G$.

3.3.3 Strumenti utilizzati

Per la validazione verranno impiegati gli strumenti definiti per le *attività*$_G$ di verifica, scelti in base alle esigenze del *progetto*$_G$ e alle specifiche del *software*$_G$.

3.4 Gestione della configurazione

3.4.1 Introduzione

La gestione della configurazione è un *processo*$_G$ che norma il tracciamento e il controllo delle modifiche a documenti e prodotti del *software*$_G$. È fondamentale nell'ambito di sviluppo *software*$_G$ eviene applicata a qualsiasi "*artefatto*$_G$". Assicura funzionamento del *sistema*$_G$ nonostante modifiche che potrebbero essere apportate.

Secondo lo *standard*$_G$ ISO/IEC12207:1995 è un processo di identificazione, organizzazione e controllo delle modifiche apportate agli artefatti. Secondo lo *standard*$_G$ IEEE 828-2018, invece, viene definito come "un processo disciplinato per gestire l'evoluzione del *software*$_G$".

3.4.2 Versionamento

Per identificare la versione di un documento si usa il formato X.Y.Z, dove:

- **X:** incrementato al raggiungimento di RTB$_G$, PB$_G$ ed eventualmente CA$_G$;
- **Y:** incrementato quando vengono apportate modifiche significative al documento. Questi possono essere cambiamenti strutturali, nuove sezioni o modifiche sostanziali;
- **Z:** incrementato per modifiche minori al documento o aggiornamenti, come correzioni di errori, miglioramenti minimi o l'aggiunta di contenuti non particolarmente rilevanti.

Ogni variazione deve essere presente nel registro delle modifiche. L'incremento dei valori più significativi porta quelli minori a zero. Ad esempio nel caso di un documento alla versione 0.7.5, qualora venisse raggiunta la *RTB*$_G$ la versione verrebbe incrementata a 1.0.0.

3.4.3 Repository

Il gruppo utilizza **GitHub**, una piattaforma basata sullo strumento di controllo versione distribuita *Git*$_G$.
Le *repository*$_G$ del gruppo ***Byte Your Dreams*** sono le seguenti:

- DocsSource: *repository*$_G$ privata per la gestione, lo sviluppo e il versionamento dei file sorgente della documentazione;
- Documents: *repository*$_G$ destinata ai committenti/proponente, che contiene i file in formato PDF riguardanti la documentazione, ottenuti dalla compilazione automatizzata dei file in formato TeX della *repository*$_G$ DocsSource;
- Proof-of-Concept: *repository*$_G$ destinata al *PoC*$_G$.

All'interno della *repository*$_G$ viene utilizzato **Gitflow** come stile di flusso di lavoro.

3.4.3.1 Flusso generale Gitflow

- Branch `main`: *branch*$_G$ principale della *repository*$_G$;
- Branch `develop`: *branch*$_G$ utilizzato per lo sviluppo, creato a partire dal *branch*$_G$ `main`;
- Branch `feature`: *branch*$_G$ utilizzato per lo sviluppo di nuove funzionalità/miglioramenti, creato a partire dal *branch*$_G$ `develop`;
- Branch `release`: *branch*$_G$ che prepara il *software*$_G$ per il rilascio, creato da *branch*$_G$ `develop`;
- Branch `hotfix`: *branch*$_G$ creato dal `main` se si verificano dei problemi;
- Merge di `develop` in `main`: fusione del *branch*$_G$ `develop` in quello `main`. Viene effettuato quando le funzionalità completate sono state verificate e validate.
- Merge di `feature` in `develop`: fusione del *branch*$_G$ `feature` in quello `develop`. Viene effettuato quando la funzionalità viene completata.
- Merge di `release` in `develop` e `main`: a seguito del completamento del *branch*$_G$ `release`, questo viene unito sul *branch*$_G$ `develop` e `main`, segnando un rilascio stabile;
- Merge di `hofix` in `develop` e `main`: a seguito della risoluzione del problema, il *branch*$_G$ `hotfix` viene fuso con i *branch*$_G$ `develop` e `main`.

Comandi

- Inizializzazione Gitflow:

```
git flow init
```

- Sviluppo di una feature:

```
git flow feature start nomeFeature
```

```
git flow feature finish nomeFeature
```

- Rilascio di una versione:

```
git flow release start X.Y.Z
```

```
git flow release finish X.Y.Z
```

- Risoluzione di un bug:

```
git flow hotfix start nomeHotfix
```

```
git flow hotfix finish nomeHotfix
```

- Pubblicazione modifiche:

```
git push origin develop
git push origin master
git push origin feature/nomeFeature
git push origin release/X.Y.Z
git push origin release/nomeHotfix
```

3.4.4 Struttura repository "Documents"

La *repository*$_G$ "Documents" è organizzata nel seguente modo:

- **Documenti Esterni:**
  - **Verbali:** contiene tutti i verbali redatti dall'inizio del *progetto*$_G$, che vedono la presenza di figure esterne;
  - **Analisi dei requisiti v2.0.0**;
  - **Manuale utente v1.0.0**;
  - **Valutazione dei costi e assunzione impegni**.
  - **Piano di qualifica v2.0.0**;
  - **Piano di progetto v2.0.0**;
  - **Specifica tecnica v1.0.0**.
- **Documenti interni:**
  - **Verbali:** contiene tutti i verbali redatti dall'inizio del *progetto*$_G$, che vedono la presenza dei soli membri del gruppo;
  - **Valutazione dei capitolati**;
  - **Glossario v2.0.0**;
  - **Norme di progetto v2.0.0**;
- Lettera di presentazione
  - di Candidatura
  - *RTB*$_G$
  - *PB*$_G$

3.4.5 Sincronizzazione e Branching

3.4.5.1 Documentazione

Per il *processo*$_G$ di documentazione, viene creato un *branch*$_G$ per ogni documento/sezione da redigere. Questo nel caso in cui due membri lavorano allo stesso documento, permette di non andare incontro a conflitti.

Convenzione per la nomenclatura dei branch relativi alla redazione/modifica di documenti

- Deve essere presente l'identificativo del documento da redarre/modificare. Di seguito è riportata una tabella che descrivere gli id:

Documento	Id
VerbaleEsterno	VE
VerbaleInterno	VI
Norme di progetto	NP
Analisi dei requisiti	AR
Piano di progetto	PP
Piano di qualifica	PQ

- Per i verbali, si mette un underscore e la data dopo l'ID: `IdDocumento_AAAAMMGG`
- Per le sezione dei documenti il norme deve essere così composto: `IdDocumento_NomeSezione`

3.5 Risoluzione dei problemi

3.5.1 Introduzione

Per analizzare e sanare le problematiche riscontrate durante lo sviluppo del *software*$_G$, viene messo in atto il *processo*$_G$ di risoluzione dei problemi, il quale ha come obiettivo quello di garantire che tali problemi individuati vengano gestiti e risolti il prima possibile.
Durante questo *processo*$_G$ vengono analizzate le problematiche emerse durante il ciclo di vita del *software*$_G$, identificandone le cause, in modo da poter prevenire rischi futuri. L'obiettivo è anche quello di garantire un miglioramento costante del prodotto che viene sviluppato, ottimizzandone i *processi*$_G$.
Il *processo*$_G$ di risoluzione dei problemi prevede attività strutturate come l'analisi delle cause, la valutazione degli impatti dei problemi riscontrati, e la definizione di azioni correttive future.
E' necessario che vengano documentati tutti i problemi riscontrati e le relative soluzioni in modo da garantire la tracciabilità degli stessi per avere un controllo nel tempo.

3.5.2 Gestione dei rischi

I rischi legati al progetto sono stati identificati e inseriti nella sezione "Analisi dei rischi" del documento *Piano di Progetto*. In questa sezione ad ogni rischio viene associata la sua probabilità di occorrenza e la misure da prendere per la loro prevenzione.
Vengono definite le strategie da mettere in atto per mitigare l'impatto che tali rischi possono avere nell'esecuzione dei vari *processi*$_G$ del *progetto*$_G$.

3.5.2.1 Metriche

Metrica	Nome
M11RNP	Rischi non previsti

Table 1: Metriche relative alla gestione dei processi

3.6 Gestione della qualità

3.6.1 Introduzione

L'obiettivo delle *attività*$_G$ eseguite durante il *processo*$_G$ di gestione della qualità è quello di garantire la qualità dei prodotti sviluppati dai *fornitori*$_G$ e di essere in grado di soddisfare e rispettare i requisiti specificati dal *proponente*$_G$. Tra le attività da svolgere bisogna identificare le metriche e i criteri di qualità, pianificare le azioni da mettere in atto per garantire e controllare la qualità, e verificare costantemente attraverso revisioni e *test*$_G$ il prodotto che si sta sviluppando. La gestione della qualità mira a garantire che il prodotto *software*$_G$ è conforme alle aspettative degli *utenti*$_G$ e ai requisiti del *progetto*$_G$.

La gestione della qualità è quindi un *processo*$_G$ che segue l'intero ciclo di vita del *software*$_G$ con l'obiettivo che il prodotto finale rispetti gli *standard*$_G$ di qualità predefiniti.

3.6.2 Attività

Le *attività*$_G$ che il *team*$_G$ si impegna a svolgere per garantire la qualità dei *processi*$_G$ e dei prodotti *software*$_G$ sviluppati sono le seguenti:

1. **Definizione degli Standard di qualità:**

   il *processo*$_G$ di gestione della qualità inizia con la definizione degli *standard*$_G$ di qualità che il *software*$_G$ deve raggiungere. Questi standard possono includere requisiti funzionali e non funzionali;

2. **Pianificazione della qualità:**

   In un piano di qualità vengono identificate *attività*$_G$ e risorse che garantiscono la qualità del prodotto durante il ciclo di vita del *progetto*$_G$;

3. **Assicurazione della qualità:**

   Durante il ciclo di vita del *progetto*$_G$ vengono svolte *attività*$_G$ di monitoraggio per garantire che i *processi*$_G$ siano conformi agli *standard*$_G$ di qualità predefiniti;

4. **Controllo della qualità:**

   il controllo della qualità prevede che vengano effettuati dei *test*$_G$ che mirano a garantire che il prodotto *software*$_G$ sviluppato è conforme agli *standard*$_G$ di qualità, e risponde ai requisiti richiesti;

5. **Gestione delle modifiche:**

   viene fatto un controllo sulle modifiche attraverso un registro delle modifiche che ha come obiettivo quello di controllare l'evoluzione di un prodotto. Questo garantisce che la qualità venga gestita anche attraverso le modifiche che vengono effettuate;

6. **Miglioramento continuo e correzione:**

   attraverso la gestione della qualità viene fatto un monitoraggio continuo durante il ciclo di vita del prodotto *software*$_G$ per permettere un miglioramento costante dei *processi*$_G$;

7. **Coinvolgimento degli stakeholder:**

   durante il *processo*$_G$ di gestione della qualità vengono coinvolti anche gli *stakeholder*$_G$ che, inviando *feedback*$_G$, aiutano a controllare che il prodotto rispetti le aspettative;

8. **Formazione e competenza del team:**

   per garantire un prodotto *software*$_G$ di qualità, il team deve costantemente studiare le tecnologie e le norme da mettere in pratica per poter rispondere alle esigenze degli *stakeholder*$_G$.

3.6.3 Piano di qualifica

Per effettuare il controllo della qualità, le *attività*$_G$ di pianificazione della stessa vengono descritte nel documento *Piano di Qualifica*, il quale è di notevole importanza nel *processo*$_G$ di gestione della qualità. In questo documento vengono definite in dettaglio le specifiche di qualità relative al prodotto *software*$_G$.

3.6.4 PDCA

Nell'ambito dell'*attività*$_G$ di miglioramento continuo e correzione, si è scelto di adottare il ciclo PDCA, conosciuto anche come ciclo di Deming.
Il ciclo PDCA è una metodologia iterativa che consente il controllo e il miglioramento continuo dei *processi*$_G$ e dei prodotti. Affinché si possa ottenere un miglioramento effettivo, bisogna attuare le seguenti 4 fasi:

- **Plan**
  Consiste nello svolgere le *attività*$_G$ di pianificazione necessarie per stabilire quali *processi*$_G$ debbano essere avviati e in quale sequenza, al fine di raggiungere obiettivi specifici;
- **Do**
  Consiste nell'effettiva esecuzione di quanto pianificato, raccogliendo dati e misurando i risultati durante lo svolgimento delle *attività*$_G$;
- **Check**
  Consiste nell'analisi e nell'interpretazione dei dati raccolti durante l'esecuzione (Do). Questi dati vengono valutati utilizzando metriche prestabilite e confrontati con gli obiettivi previsti nella fase di Plan.
- **Act**
  Si consolida quanto di positivo è stato rilevato nella fase precedente (Check) e si implementano le strategie correttive necessarie per migliorare gli aspetti che non hanno raggiunto i risultati attesi, analizzandone approfonditamente le cause. In questo modo, il ciclo PDCA viene continuamente perfezionato e ogni iterazione successiva aggiunge valore al *processo*$_G$.

3.6.5 Strumenti

Gli strumenti utilizzati nel *processo*$_G$ di gestione della qualità sono rappresentati dalle metriche.

3.6.6 Struttura e identificazione metriche

- **Metrica:**
  identificativo della metrica nel formato:

  M [numero] [abbreviazione]

  dove:

  - `M`: sta per metrica;
  - `[numero]`: numero progressivo univoco per ogni metrica;
  - `[abbreviazione]`: abbreviazione composta dalle iniziali del nome della metrica.
- **Nome:** specifica il nome della metrica;
- **Descrizione:** breve descrizione della funzionalità della metrica adottata;

3.6.7 Criteri di accettazione

Per ciascuna metrica elencata nel documento *Piano di Qualififca* vengono definiti in formato tabellare:

- **Valore di accettazione:** valore che la metrica deve raggiungere per essere considerata soddisfacente o conforme agli *standard*$_G$ stabiliti;
- **Valore preferibile:** valore ideale che dovrebbe essere assunto dalla metrica.

3.6.8 Metriche

Metrica	Nome
M1PMS	Percentuale di Metriche Soddisfatte (PMS)
M24DE	Densità degli Errori (DE)

Table 2: Metriche relative alla gestione dei processi

4 Processi organizzativi

Lo sviluppo di *software*$_G$ è un campo articolato e multidisciplinare che necessita di una pianificazione meticolosa, una gestione ottimale delle risorse e un controllo stringente della qualità. L'implementazione di processi organizzativi ben definiti diventa quindi essenziale per assicurare il successo del *progetto*$_G$ in questione. È fondamentale adottare metodologie strutturate e strumenti adeguati per coordinare le attività, monitorare i progressi e garantire che ogni fase del progetto rispetti gli standard di qualità previsti. Solo attraverso un'attenta pianificazione e una gestione efficace delle risorse è possibile raggiungere gli obiettivi prefissati e consegnare un prodotto finale che soddisfi le aspettative degli utenti.

4.1 Gestione dei Processi

4.1.1 Introduzione

La Gestione dei Processi si dedica a definire, attuare e ottimizzare i processi che orientano lo sviluppo del *software*$_G$, con l'obiettivo di raggiungere i traguardi stabiliti e rispondere alle aspettative degli

stakeholder$_G$. Questo ambito include la creazione di procedure ben strutturate, il monitoraggio continuo delle attività e l'adozione di miglioramenti per garantire l'efficienza e la qualità del prodotto finale. È essenziale che ogni fase del processo sia attentamente pianificata e gestita per assicurare che le esigenze degli utenti e degli *stakeholder*$_G$ siano pienamente soddisfatte.

4.1.1.1 Attività di gestione di processo

Le *attività*$_G$ di Gestione dei Processi includono:

1. **Definizione dei Processi:**
    - Riconoscere e registrare i *processi*$_G$ chiave coinvolti nello sviluppo del *software*$_G$.
    - Stabilire direttive e procedure per l'implementazione di ogni *processo*$_G$.
2. **Programmazione e Supervisione:**
    - Creare piani dettagliati per l'implementazione dei *processi*$_G$.
    - Sorvegliare costantemente il progresso, l'efficacia e la conformità ai requisiti stabiliti.
    - Valutare tempi, risorse e costi necessari.
3. **Analisi e Ottimizzazione costante:**
    - Effettuare valutazioni periodiche dei *processi*$_G$ per individuare aree di miglioramento.
    - Attuare azioni di correzione e di preventivaggio per ottimizzare i *processi*$_G$.
4. **Formazione e competenze:**
    - Garantire che il personale coinvolto nei *processi*$_G$ sia adeguatamente istruito.
    - Conservare e potenziare le competenze indispensabili per una gestione efficiente dei *processi*$_G$.
5. **Gestione dei rischi:**
    - Riconoscere e analizzare i rischi legati ai *processi*$_G$.
    - Stabilire piani per ridurre o affrontare i rischi individuati.

4.1.2 Pianificazione

4.1.2.1 Descrizione

La pianificazione è cruciale nella gestione dei *processi*$_G$, poiché si propone di sviluppare un piano strutturato e coerente per garantire un'efficace realizzazione delle *attività*$_G$ lungo tutto il ciclo di vita del *software*$_G$.
Il responsabile del gruppo ha il compito fondamentale di orchestrare ogni aspetto riguardante la pianificazione delle *attività*$_G$, che comprende l'assegnazione delle risorse, la determinazione delle tempistiche e la stesura di piani dettagliati. Inoltre, il responsabile deve assicurarsi che ciascun membro del team possa eseguire con piena fattibilità ed efficienza il piano elaborato da lui stesso.
Di rilevante importanza è la presenza delle risorse necessarie, in termini di tempo, tecnologie e personale, per lo svolgimento di ciascuna *attività*$_G$.

4.1.2.2 Obiettivi

La pianificazione ha come ruolo principale la rotazione dei ruoli all'interno del gruppo, garantendo così che ciascun membro possa assumere, almeno una volta, ciascun ruolo nell'arco della durata del *progetto*$_G$. Lo scopo di tale obiettivo è quello di distribuire equa responsabilità all'interno del team, in modo tale che ciascun membro possa avere una diversa prospettiva dello stesso *progetto*$_G$, conseguendo così un aumento personale delle competenze, necessario per un avanzamento lineare e corretto.

4.1.2.3 Assegnazione dei ruoli

Nell'arco di svolgimento del *progetto*$_G$ i ruoli che ciascun membro del gruppo assumerà, sono i seguenti:

1. Responsabile
2. Amministratore
3. Analista
4. Progettista
5. Verificatore
6. Programmatore

4.1.2.4 Responsabile

Persona chiave nell'avanzamento del *progetto*$_G$ in quanto coordina il lavoro del gruppo, fungendo come mediatore tra la *committente*$_G$ ed il gruppo stesso.
Nello specifico, si occupa di programmare le *attività*$_G$ stabilendo quali svolgere, le tempistiche, e l'assegnazione delle priorità. Inoltre, valuta i rischi delle decisioni da prendere, monitora i progressi del progetto, gestisce il personale e approva la documentazione.

4.1.2.5 Amministratore

L'amministratore è una figura professionale, il cui obiettivo è quello di supervisionare e gestire l'ambiente di lavoro utilizzato dal gruppo e anche quello di occuparsi della documentazione relativa alle norme di progetto.

4.1.2.6 Analista

Figura professionale di rilevante importanza all'interno del gruppo, in quanto capace di applicare uno studio approfondito alle esigenze e specifiche del *progetto*$_G$, reinterpretandole in requisiti precisi e completi, indispensabili per tracciare le linee guida utili per un corretto avanzamento del *progetto*$_G$.

4.1.2.7 Progettista

Il progettista è una figura professionale che svolge un ruolo cruciale all'interno di un team di sviluppo *software*$_G$. Esso si occupa di tutte le scelte progettuali derivati dalla stesura dei requisiti applicando, quindi, tecnologie e scelte architetturali più inerenti alla casistica espressa dal lavoro dell'analista.
Il progettista non si occupa della manutenzione del prodotto.

4.1.2.8 Programmatore

Il programmatore è quella figura professionale adibita alla stesura del codice *software*$_G$. Questa figura deve implementare il codice seguendo i requisiti previsti dall'analista ed anche l'*architettura*$_G$ idealizzata dal progettista.
Inoltre, il programmatore ha il compito di verificare e validare il codice scritto da se stesso, garantendo qualità e correttezza del prodotto finale.

4.1.2.9 Verificatore

Lo scopo di questa figura professionale è quello di prendere atto del lavoro svolto dagli altri membri del gruppo per constatarne l'effettiva correttezza relativa alla qualità e alla conformità delle attese prefissate. Esso deve, quindi, verificare la conformità dei prodotti alle *Norme di Progetto* ed inoltre, verificare la conformità dei prodotti ai requisti funzionali e di qualità stilati dagli analisti, segnalando tutti gli eventuali errori riscontrati nell'*attività*$_G$ di verifica.

4.1.2.10 Ticketing

Per tracciare i diversi punti di avanzamento del *progetto*$_G$ viene utilizzato *GitHub*, un *sistema*$_G$ appositamente ideato per il tracciamento delle *issue*$_G$, dando così la possibilità di una gestione pratica e chiara delle diverse *attività*$_G$ da completare.
La gestione delle *issue*$_G$ viene delegata prettamente all'amministratore del gruppo, che in base alle *attività*$_G$ rilevate dal responsabile, le crea e le assegna ai membri del gruppo.
Dato l'utilizzo di questo *sistema*$_G$ di tracciamento, ogni membro del gruppo, grazie alla presenza di strumenti specifici come la *DashBoard* e la *Roadmap*, ha la garanzia di avere una visione completa dello stato di avanzamento delle diverse *issue*$_G$.

Iter di generazione delle issue
Le *issue*$_G$ vengono gestite interamente dall'amminstratore del gruppo, il quale deve provvedere a corredarle di specifici attributi, quali:

- **Titolo:** deve essere sintetico e chiaro.
- **Descrizione:** breve e coincisa descrizione della *issue*$_G$.
- **Incaricato:** incaricato allo svolgimento delle *issue*$_G$. Possono esserci più incaricati per *issue*$_G$.
- **Labels:** tag utilizzato per rendere nota la categoria di apparteneza della *issue*$_G$. I *label* utilizzati sono:
  - **NP:** Norme di Progetto.
  - **PP:** Piano di Progetto.
  - **PQ:** Piano di Qualifica.
  - **AR:** Analisi dei Requisiti.
  - **Poc:** Proof of concept.
- **Milestone:** *milestone*$_G$ associata alla *issue*$_G$.
- **Projects**: progetto a cui la *issue*$_G$ è associata.
  La presenza di una *DashBoard*$_G$ associata ad un progetto, denota la presentazione, al suo interno, delle diverse *issue*$_G$ associate a tale progetto.
- **Development:** *branch*$_G$ e Pull Request associate alla *issue*$_G$.
  All'accetazione di una Pull Request ne consegue la chiusura della relativa *issue*$_G$ ed il suo spostamento nella colonna "Done" della *DashBoard*$_G$.

Ciclo di vita di una issue
Durante il suo ciclo di vita, una *issue*$_G$ si sofferma, per un limitato periodo di tempo, in ciascun possibile stato che una *issue*$_G$ può assumere. Questi stati sono:

- **Backlog:** in questa fase, le *issue*$_G$ vengono create dall'amministratore e raccolte. Le *issue*$_G$ nel "Backlog" rappresentano *attività*$_G$ che devono essere affrontate ma non ancora pronte per essere lavorate.
- **Ready:** una volta che una *issue*$_G$ è stata valutata e prioritizzata, viene posta nella fase "Ready". Questo significa che la*issue*$_G$ è pronta per essere iniziata dal membro/i del team incaricato/i.
- **In Progress:** quando l'incaricato inizia a lavorare sulla *issue*$_G$ assegnata, questa viene posta nella fase "In Progress". In questa fase, la *issue*$_G$ è effettivamente in lavorazione e il suo stato di avanzamento può essere monitorato.
- **In Review:** una volta che la *issue*$_G$ viene considerata terminata, l'incaricato apre una Pull Request su GitHub e all'interno della *DashBoard*$_G$, la *issue*$_G$ deve essere spostata nella fase "In Review", pronta per essere verificata da un verificatore.
- **Done:** se la *issue*$_G$ supera la fase di verifica realizzata dal verificatore, allora essa viene trasferita dalla colonna "In Review" alla colonna "Done", e nel caso di associazione della *issue*$_G$ a una Pull Request, una volta che quest'ultima viene confermata dal verificatore, la *issue*$_G$ viene automaticamente chiusa e spostata nella colonna "Done" della *DashBoard*$_G$

4.1.2.11 Strumenti

- **GitHub:** *sistema*$_G$ utilizzato per il tracciamento e la gestione dell'intero ciclo di vita delle *issue*$_G$.

4.1.3 Coordinamento

4.1.3.1 Descrizione

Con coordinamento viene intesa l'*attività*$_G$ che supervisiona la gestione della comunicazione, sia interna, ovvero tra i membri del gruppo, sia esterna, ovvero con il *proponente*$_G$ e i *committenti*$_G$, in modo tale che tutte le parti che influiscono sullo sviluppo del *progetto*$_G$ possano essere partecipi al suo avanzamento.

4.1.3.2 Obiettivi

L'obiettivo principale del coordinamento è assicurare efficienza nello sviluppo del prodotto *software*$_G$, prevenendo potenziali problemi quali ritardi e malintesi, e concedendo la possibilità a tutti coloro che attivamente partecipano all'avanzamento del *progetto*$_G$, di avere una visione completa e precisa di esso. Una chiara conseguenza del coordinamento è un rafforzamento della coesione dei membri del gruppo, i quali attraverso uno scambio di pareri e punti di vista differenti, possono essere in grado di risolvere agevolmente le diverse problematiche che possono insorgere durante l'avanzamento.

Comunicazione
Per quanto riguarda la comunicazione, il team *Byte Your Dreams* mantiene comunicazioni costanti, sia interne che esterne del gruppo, che possono essere sincrone o asincrone, in base alle esigenze.

4.1.3.3 Comunicazioni sincrone

- **Comunicazioni sincrone interne**
  Per quanto riguarda le comunicazioni sincrone interne , il gruppo *Byte Your Dreams* ha deciso di utilizzare *Discord*$_G$ dato che questo servizio è gratuito e permette di comunicare sia attraverso chiamate vocali, videochiamate, messaggi di testo, media, file in chat private, oltre che consentire una comoda utilità di condivisione schermo.
- **Comunicazioni sincrone esterne**
  Per quanto riguarda invece le comunicazioni sincrone esterne è stato scelto, in comune accordo con l'azienda, di utilizzare *Teams*$_G$.

4.1.3.4 Comunicazioni asincrone

- **Comunicazioni asincrone interne**
  Per le comunicazioni asincrone interne viene utilizzata l'applicazione *WhatsApp*$_G$ all'interno di un gruppo dedicato.
- **Comunicazioni asincrone esterne**
  Per le comunicazioni asincrone esterne, è stata presa la decisione, in concomitanza con l'azienda, di utilizzare *Gmail*$_G$, un servizio di posta elettronica gratuito offerto da Google oppure il servizio di messaggistica messo a disposizione da *Teams*$_G$.

Riunioni In ogni riunione tenuta dal gruppo viene scelto un segretario con il compito di prendere appunti, attraverso i quali, successivamente, redigere un verbale che contenga tutti i punti chiave trattati durante il meeting.

4.1.3.5 Riunioni interne

È stato scelto di tenere riunioni interne con frequenza settimanale, allo lo scopo di aggiornare tutti i componenti del *team*$_G$ sugli avanzamenti avvenuti durante l'arco settimanale dando al team una visione complessiva dello stato del *progetto*$_G$.
Solitamente le riunioni interne avvengono il lunedì o il martedì, in base agli impegni di ciascun membro del gruppo; la maggior parte delle riunioni verranno tenute nella fascia pomeridiana, in quanto la mattina verrà dedicata alle lezioni universitarie.

In caso di assenza di uno dei membri del gruppo, la riunione può essere riprogrammata a quando la disponibilità dei componenti è completa; il tutto avviene tramite una comunicazione interna dei membri del gruppo tramite canale *WhatsApp*$_G$.
Ciascun membro del gruppo ha la libertà di richiedere una riunione ulteriore ogniqualvolta sia necessario; tale riunione verrà organizzata tramite comunicazione interna attraverso *WhatsApp*$_G$.
Le riunioni interne ricoprono un ruolo di elevata importanza per lo svolgimento del *progetto*$_G$ in quanto permettono a ciascun membro del gruppo di monitorare gli avanzamenti di ciascuna *attività*$_G$ avendo così la possibilità di avere una panoramica completa di esso. Non in secondo luogo, le riunioni interne hanno lo scopo di favorire il dialogo e lo scambio di idee all'interno del *team*$_G$ con l'obbiettivo di facilitare la risoluzione di possibili problemi e per la costruzione di un ambiente sereno e favorevole alla comunicazione e alla condivisione.
Le riunioni interne vengono tenute attraverso il canale di comunicazione *Discord*$_G$ per facilitare la comunicazione interna sincrona.
Nelle riunioni interne, il responsabile del gruppo svolge un ruolo di rilevante importanza in quanto:

- Prepara in anticipo gli argomenti da trattare durante la riunione.
- Dà le linee guida delle conversazioni e raccoglie i pareri e punti di vista di ciascun membro.
- Sceglie preventivamente un segretario per la raccolta degli appunti durante lo svolgimento della riunione.
- Al termine della riunione ha il compito di proporre e assegnare le nuove *attività*$_G$ da svolgere in prospettiva della futura riunione.

Verbali interni
Le riunioni interne sono così strutturate:

- **Parte iniziale:** viene fatta mente locale sul periodo trascorso dall'ultima riunione interna, cercando di elencare tutti i possibili punti problematici riscontrati e dando, a tutti i membri, lo stato di avanzamento di tutte le precedenti *attività*$_G$.
- **Parte centrale:** vengono discussi tutti i punti stilati nell'ordine del giorno.
- **Parte finale:** vengono proposte le nuove *attività*$_G$ da svolgere.

Alla conclusione di ogni riunione, l'amministratore associa una *issue*$_G$ in GitHub, relativa all'incarico di redigere il verbale interno, al segretario scelto precedentemente alla riunione.

4.1.3.6 Riunioni esterne

Le riunioni esterne sono essenziali per una fluida continuazione del *progetto*$_G$, in quanto un incontro esterno con l'azienda può risolvere possibili dubbi da parte dei membri.
Il responsabile svolge un ruolo di rilevanza poichè ha il compito di programmare tali incontri in concomitanza con l'azienda e di occuparsi del loro svolgimento garantendo efficienza ed efficacia.
Durante le riunioni esterne sarà il responsabile ad esporre i punti di discussione al *proponente*$_G$, assicurando così una comunicazione ottimale fra le diverse parti e gestendo efficacemente il tempo a disposizione.
I membri del gruppo si sforzano il più possibile per garantire la loro partecipazione agli incontri. Nel caso di impossibilità, dovranno avvertire il prima possibile il responsabile, il quale, a sua volta, provvederà ad aggiornare l'azienda sulla situazione, richiedendo uno slittamento dell'incontro.

Riunione con l'azienda proponente
In concomitanza con l'azienda, si è deciso di tenere un *SAL*$_G$ (stato di avanzamento lavori) con frequenza bisettimanale tramite il canale di comunicazione *Teams*$_G$.
La scaletta della riunione esterna è la seguente:

- **Revisione del periodo precedente:** viene fatto il punto della situazione su quanto fatto dall'ultimo *SAL*$_G$ mettendo in evidenza i punti risolti ed i punti non risolti.

- **Ordine del giorno:** principali punti di discussione della riunione, di particolare interesse per i membri del gruppo e indispensabili per l'avanzamento del *progetto*$_G$.
- **Chiarimenti ulteriori:** ulteriori punti di discussione che solitamente conseguono punti di discussione relativi all' "Ordine del giorno". Dubbi che derivano da altri dubbi maggiori oppure dubbi di interesse secondario.

Verbali esterni
I verbali esterni vengono redatti dal segretario e successivamente, il responsabile, si occupa della condivisione di esso con l'azienda *proponente*$_G$ tramite *Google Drive*$_G$.

4.1.3.7 Strumenti

- **Discord:** utilizzato per la comunicazione sincrona e per le riunioni interne.
- **WhatsApp:** utilizzato per la comunicazione asincorona con i membri del gruppo.
- **Teams:** utilizzato per le riunioni esterne e per la comunicazione asincora con l'azienda *proponente*$_G$.
- **Gmail:** *servizio*$_G$ di posta elettronica utilizzato per la comunicazione asincrona con l'azienda.
- **Google Drive:** *servizio*$_G$ di condivisione di documenti messo a disposizione da Google ed utilizzato dal gruppo per la condivisione di verbali esterni con l'azienda *proponente*$_G$.

4.1.3.8 Metriche

Metrica	Nome
M6SV	Schedule Variance (SV)
M21IF	Implementazione delle Funzionalità (IF)
M4BV	Budget Variance (BV)
M12VR	Variazione dei Requisiti (VR)

Table 3: Metriche relative alla gestione dei processi

5 Standard per la qualità

Durante la valutazione della qualità dei *processi*$_G$ e del *software*$_G$ si è fatto affidamento a degli *standard*$_G$ internazionali sviluppati per fornire una struttura coerente e standardizzata per identificare, misurare e valutare la qualità di un prodotto *software*$_G$. Lo standard ISO/IEC 9126 viene usato per gestire parametri come funzionalità, affidabilità, usabilità, efficienza, manutenibilità e portabilità. Questa suddivisione garantisce di avere un controllo completo sulla qualità del prodotto *software*$_G$.
Lo *standard*$_G$ ISO/IEC 12207:1995 invece suddivide i *processi*$_G$ software in primari,di supporto e organizzativi; è progettato per fornire una struttura che descriva le *attività*$_G$ e i compiti necessari per sviluppare, utilizzare e mantenere un *sistema*$_G$ software.
Infine, lo standard ISO/IEC 25010 fornisce un *framework*$_G$ che permette di individuare e misurare le metriche di qualità del *software*$_G$.

5.1 Caratteristiche del Sistema, Standard ISO/IEC 25010

5.1.1 Funzionalità

- **Adeguatezza funzionale:** Copertura completa di tutte le funzionalità richieste;

- **Accuratezza:** Capacità del *software*$_G$ di fornire risultati corretti e precisi;
- **Interoperabilità:** Capacità del *software*$_G$ di interagire con altri *sistemi*$_G$.

5.1.2 Affidabilità

- **Maturità:** Capacità del *software*$_G$ di evitare errori o guasti in condizioni standard;
- **Tolleranza agli Errori:** Capacità del *software*$_G$ di continuare a funzionare anche in presenza di guasti;
- **Recuperabilità:** Capacità del *software*$_G$ di ripristinarsi dopo un errore.

5.1.3 Usabilità

- **Comprensibilità:** Facilità con cui gli *utenti*$_G$ possono comprendere il *software*$_G$;
- **Apprendibilità:** Facilità con cui un *utente*$_G$ può imparare a utilizzare il *software*$_G$;
- **Operabilità:** Facilità d'uso durante l'operatività del *software*$_G$;

5.1.4 Efficienza

- **Tempo di Risposta:** Tempo impiegato dal *software*$_G$ per rispondere alle richieste dell'*utente*$_G$;
- **Utilizzo delle Risorse:** Efficienza nell'uso delle risorse del *sistema*$_G$.
- **Capacità:** Capacità del *software*$_G$ di gestire quantità crescenti di *utenti*$_G$ o *dati*$_G$.

5.1.5 Manutenibilità

- **Analizzabilità:** Facilità con cui è possibile diagnosticare guasti o identificare le parti da modificare;
- **Modificabilità:** Capacità di apportare modifiche al *software*$_G$ senza difficoltà;
- **Stabilità:** Capacità del *software*$_G$ di funzionare correttamente dopo le modifiche;

5.1.6 Portabilità

- **Adattabilità:** Capacità del *software*$_G$ di essere adattato a diversi ambienti senza modifiche estese;
- **Installabilità:** Facilità con cui il *software*$_G$ può essere installato in un ambiente specifico;
- **Conformità:** il rispetto delle norme e degli *standard*$_G$ relativi alla portabilità.

5.2 Suddivisione secondo Standard ISO/IEC 12207:1995

5.2.1 Processi primari

Sono gli elementi fondamentali legati al ciclo di vita del *software*$_G$, che includono lo sviluppo, l'implementazione e la gestione del *software*$_G$. Questi *processi*$_G$ sono al centro del modello e sono progettati per guidare le organizzazioni nella creazione, acquisizione e manutenzione dei *sistemi*$_G$ software.

5.2.2 Processi di supporto

I *processi*$_G$ di supporto forniscono strumenti e attività per assistere i processi primari e organizzativi, contribuendo a garantire la qualità del *software*$_G$ e il corretto svolgimento delle *attività*$_G$. Questi *processi*$_G$ non producono direttamente il *software*$_G$, ma sono fondamentali per assicurare che il prodotto finale soddisfi i requisiti e sia conforme agli *standard*$_G$.

5.2.3 Processi organizzativi

I *processi*$_G$ organizzativi sono progettati per supportare la gestione complessiva del ciclo di vita del *software*$_G$ a livello aziendale e assicurare che le attività siano pianificate, monitorate e migliorate in modo sistematico. Questi *processi*$_G$ si concentrano sugli aspetti strategici e gestionali, fornendo un quadro di riferimento per una continua ottimizzazione.

6 Metriche di qualità

Le metriche di qualità di un prodotto *software*$_G$ sono strumenti di misurazione impiegati per analizzare e monitorare vari aspetti della qualità del *software*$_G$ e del *processo*$_G$ di sviluppo. Il loro scopo è assicurare che il *software*$_G$ rispetti i requisiti funzionali e non funzionali evidenziando eventuali aree che necessitano miglioramenti e garantire il rispetto alle conformità dello stesso.

6.1 Metriche per la qualità di processo

- **Metrica M1PMS:**
  - **Nome:** Percentuale di Metriche Soddisfatte (PMS);
  - **Descrizione:** Misura la proporzione delle metriche definite che sono state effettivamente adottate o soddisfatte in un *progetto*$_G$;
  - **Formula:** $\frac{metriche\ soddisfatte}{metriche\ totali} \times 100$
- **Metrica M2EAC:**
  - **Nome:** Estimated at Completion (*EAC*$_G$);
  - **Descrizione:** Fornisce una stima del costo totale previsto per completare un *progetto*$_G$;
  - **Formula:** $EAC_G = \frac{BAC}{CPI}$
- **Metrica M3CPI:**
  - **Nome:** Cost Performance Index (*CPI*$_G$);
  - **Descrizione:** Misura l'efficienza dei costi di un *progetto*$_G$;
  - **Formula:** $CPI_G = \frac{EV}{AC}$
- **Metrica M4BV:**

  - **Nome:** Budget Variance (BV);
  - **Descrizione:** Calcola la differenza percentuale tra il budget pianificato e l'effettivo costo sostenuto;
  - **Formula:** $BV = EV - AC$
- **Metrica M5AC:**
  - **Nome:** Actual Cost (AC);
  - **Descrizione:** Misurazione aggiornata dei costi effettivamente sostenuti dall'inizio di un *progetto*$_G$;
  - **Formula:** Dato disponibile e aggiornato in "Piano di progetto" per ogni periodo.
- **Metrica M6SV:**
  - **Nome:** Schedule Variance (SV);
  - **Descrizione:** Misura quanto un progetto è in anticipo o in ritardo rispetto alla pianificazione temporale;
  - **Formula:** $SV = EV - PV_G$
- **Metrica M7EV:**
  - **Nome:** Earned Value (EV);
  - **Descrizione:** Misura il valore del lavoro effettivamente completato fino a una data specifica;
  - **Formula:** $EV = \%lavoro\ svolto \times EAC_G$
- **Metrica M8PV:**
  - **Nome:** Planned Value (PV_G);
  - **Descrizione:** Fornisce una stima dei costi previsti per le *stakeholder*$_G$ future di un *progetto*$_G$;
  - **Formula:** $PV_G = \%lavoro\ svolto \times BAC$
- **Metrica M9ETC:**
  - **Nome:** Estimate to Complete (ETC_G);
  - **Descrizione:** Fornisce una stima dei costi necessari per completare un *progetto*$_G$, basandosi sul lavoro rimanente e sulle spese previste;
  - **Formula:** $ETC_G = EAC_G - AC$
- **Metrica M11RNP:**
  - **Nome:** Rischi Non Previsti (RNP);
  - **Descrizione:** Fornisce una misurazione della capacità di un *progetto*$_G$ di evitare l'emergere di rischi non identificati in fase di pianificazione;
- **Metrica M12VR:**
  - **Nome:** Variazione dei Requisiti (VR);
  - **Descrizione:** Metrica della stabilità dei requisiti di un *progetto*$_G$, indicando quanto sono stati soggetti a modifiche rispetto alla pianificazione iniziale;
  - **Formula:** $NRA + NRR + NRM$, dove:
    - *NRA* (Numero Requisiti Aggiunti) è la quantità di requisiti aggiunti dall'ultimo incremento;
    - *NRR* (Numero Requisiti Rimossi) è la quantità di requisiti rimossi dall'ultimo incremento;
    - *NRM* (Numero Requisiti Modificati) è la quantità di requisiti modificati dall'ultimo incremento.

- **Metrica M13PCTS:**
  - **Nome:** Percentuale di Casi di Test Superati (PCTS);
  - **Descrizione:** Fornisce una misura della qualità del *software*$_G$, indicando quanto efficacemente il software ha superato i *test*$_G$ di verifica;
  - **Formula:** $\frac{numero\ di\ casi\ di\ test\ superati}{numero\ totale\ di\ casi\ di\ test} \times 100$
- **Metrica M14PCTF:**
  - **Nome:** Percentuale di Casi di Test Falliti (PCTF);
  - **Descrizione:** Fornisce una misura della qualità del *software*$_G$, indicando quanti *test*$_G$ non sono stati superati durante la fase di verifica;
  - **Formula:** $\frac{numero\ di\ casi\ di\ test\ non\ superati}{numero\ totale\ di\ casi\ di\ test} \times 100$
- **Metrica M15SC:**
  - **Nome:** Statement Coverage (*MSC*$_G$);
  - **Descrizione:** Misura la percentuale di istruzioni eseguite almeno una volta durante i *test*$_G$;
  - **Formula:** $MSC_G = \frac{istruzioni\ eseguite}{istruzioni\ totali} \times 100$
- **Metrica M16BC:**
  - **Nome:** Branch Coverage (MBC);
  - **Descrizione:** Misura la percentuale di rami eseguiti nei punti di decisione del codice;
  - **Formula:** $MBC = \frac{flussi\ funzionali\ implementati\ e\ testati}{flussi\ condizionali\ riusciti\ e\ non} \times 100$
- **Metrica M17CNC:**
  - **Nome:** Condition Coverage (CNC);
  - **Descrizione:** Misura la percentuale di condizioni booleane valutate sia come true sia come false all'interno di ciascun punto di decisione;
  - **Formula:** $CNC = \frac{numero\ di\ operandi\ eseguiti}{numero\ totale\ di\ operandi\ eseguiti} \times 100$

6.2 Metriche per la qualità di prodotto

- **Metrica M18PROS:**
  - **Nome:** Percentuale di Requisiti Obbligatori Soddisfatti (PROS);
  - **Descrizione:** Metrica cruciale per valutare il successo di un *progetto*$_G$ software, verifica la conformità del prodotto ai requisiti essenziali definiti nell'analisi dei requisiti;
  - **Formula:** $\frac{requisiti\ obbligatori\ soddisfatti}{requisiti\ obbligatori\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M19PRDS:**
  - **Nome:** Percentuale di Requisiti Desiderati Soddisfatti (PRDS);
  - **Descrizione:** Misura l'estensione con cui i requisiti che migliorano l'esperienza dell'utente sono stati implementati, contribuendo a creare un prodotto non solo funzionale ma anche più piacevole e completo;
  - **Formula:** $\frac{requisiti\ desiderabili\ soddisfatti}{requisiti\ desiderabili\ totali} \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M20PRPS:**

- **Nome:** Percentuale di Requisiti Opzionali Soddisfatti (PRPS);
- **Descrizione:** Metrica che valuta la quantità di requisiti opzionali implementati in un prodotto *software*$_G$;
- **Formula:** $\frac{requisiti\ opzionali\ soddisfatti}{requisiti\ opzionali\ totali} \times 100$
- **Caratteristica di qualità:** Funzionalità.

- **Metrica M21IF:**
  - **Nome:** Implementazione delle Funzionalità (IF);
  - **Descrizione:** Indicatore diretto del grado di completamento dello sviluppo *software*$_G$ in relazione alle funzionalità inizialmente pianificate;
  - **Formula:**$(1 - \frac{F_{NL}}{F_L}) \times 100$
  - **Caratteristica di qualità:** Funzionalità.
- **Metrica M22CO:**
  - **Nome:** Correttezza Ortografica (CO);
  - **Descrizione:** Valuta l'assenza di errori di ortografia nella documentazione del *software*$_G$;
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M23IG:**
  - **Nome:** Indice Gulpease (MIG);
  - **Descrizione:** Valuta la leggibilità di un testo in base alla lunghezza media delle parole e delle frasi;
  - **Formula:** $IG = 89 + \frac{300 \cdot N_f - 10 \cdot N_l}{N_p}$
  - **Dove:**
    - N_f: numero di frasi;
    - N_l: numero di lettere;
    - N_p: numero di parole.
  - I risultati sono compresi tra 0 e 100, dove il valore "100" indica la leggibilità più alta e "0" la leggibilità più bassa. In generale risulta che i testi con un indice:
    - < 80: sono difficili da leggere per chi ha la licenza elementare;
    - < 60: sono difficili da leggere per chi ha la licenza media;
    - < 40: sono difficili da leggere per chi ha un diploma superiore.
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M24DE:**
  - **Nome:** Densità degli Errori (DE);
  - **Descrizione:** Metrica che calcola la percentuale di errori presenti nel codice di un *software*$_G$ rispetto alla quantità totale di codice;
  - **Formula:** $DE = \frac{numero\ di\ errori}{totale\ delle\ linee\ di\ codice} \times 100$
  - **Caratteristica di qualità:** Affidabilità.
- **Metrica M25ATC:**
  - **Nome:** Accoppiamento tra Classi (ATC);
  - **Descrizione:** Misura il grado di dipendenza e interconnessione tra le classi di un sistema *software*$_G$;
  - **Caratteristica di qualità:** Manutenibilità.

- **Metrica M26MCCM**
  - **Nome:** Complessità Ciclomatica per Metodo ($MCCM_G$);
  - **Descrizione:** Misura la complessità di un metodo in base ai possibili livelli di annidamento percorribili all'interno di esso;
  - **Formula:** $MCCM_G = e - n + 2$
  - **Dove:**
    - e: Numero di archi del grafo del flusso di esecuzione del metodo;
    - n: Numero di vertici del grafo del flusso di esecuzione del metodo.
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M27PM:**
  - **Nome:** Parametri per Metodo (PM);
  - **Descrizione:** Valore massimo di parametri per ogni metodo;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M28APC:**
  - **Nome:** Attributi Per Classe (APC);
  - **Descrizione:** Calcola l'ammontare massimo di attributi per classe;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M29LCM:**
  - **Nome:** Linee di Codice per Metodo (LCM);
  - **Descrizione:** Misura la lunghezza massima di ogni metodo dalla lunghezza del codice (in linee);
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M30PG:**
  - **Nome:** Profondità delle Gerarchie (PG);
  - **Descrizione:** Misura il numero di livelli tra una classe base e le sue sottoclassi in una gerarchia di ereditarietà;
  - **Caratteristica di qualità:** Manutenibilità.
- **Metrica M31TMR:**
  - **Nome:** Tempo Medio di Risposta (TMR);
  - **Descrizione:** Valuta quanto un sistema $software_G$ è veloce e reattivo nel rispondere alle richieste;
  - **Formula:** $\frac{somma\ dei\ tempi\ di\ risposta}{numero\ totale\ di\ misurazioni}$
  - **Caratteristica di qualità:** Efficienza.
- **Metrica M32FU:**
  - **Nome:** Facilità di Utilizzo (FU);
  - **Descrizione:** Metrica che valuta il numero di click necessari per raggiungere un obiettivo all'interno del sistema $software_G$;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M33TA:**

  - **Nome:** Tempo di Apprendimento (TA);
  - **Descrizione:** Misura il tempo massimo necessario a un utente per imparare ad usare un prodotto;
  - **Caratteristica di qualità:** Usabilità.
- **Metrica M34VBS:**
  - **Nome:** Versioni dei Browser Supportate (VBS);
  - **Descrizione:** Calcola in percentuale quante sono le versioni di browser compatibili con un sistema $software_G$ rispetto al totale delle versioni disponibili;
  - **Caratteristica di qualità:** Portabilità.