



ByteOps.swe@gmail.com

Norme di progetto

Informazioni documento

Redattori	A. Barutta R. Smanio N. Preto
Verificatori	E. Hysa L. Skenderi D. Diotto
Amministratore	F. Pozza
Destinatari	T. Vardanega R. Cardin

Registro delle modifiche

Versione	Data	Autore	Verificatore	Dettaglio
0.4.0	28/11/2023	A. Barutta	F. Pozza	Sezione Verifica
0.3.0	21/11/2023	A. Barutta	E. Hysa	Sezione Gestione della configurazione
0.2.0	06/11/2023	A. Barutta N. Preto	R. Smanio	Sezione Comunicazione Gestione incontri
0.1.0	05/11/2023	A. Barutta N. Preto	R. Smanio	Sezione Documentazione, Introduzione

Indice

ByteOps

Contents

1	Introduzione	7
1.1	Finalità 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	8
2.1	Sviluppo	8
2.1.1	Introduzione	8
2.1.2	Analisi dei requisiti	8
2.1.2.1	Obbiettivi:	8
2.1.2.2	Documentazione	8
2.1.2.3	Casi d'uso	9
2.1.2.4	Diagrammi dei casi d'uso	10
2.1.2.5	Requisiti	15
2.1.2.6	Metriche	16
2.1.2.7	Strumenti	17
2.1.3	Progettazione	17
2.1.3.1	Scopo	17
2.1.3.2	Descrizione	17

2.1.3.3	Aspettative	17
2.1.3.4	Strumenti	17
2.1.4	Codifica	18
2.1.4.1	Scopo	18
2.1.4.2	Aspettative	18
2.1.4.3	Norme di codifica	18
2.1.4.4	Commenti	19
2.1.5	Configurazione dell'ambiente di esecuzione	19
2.1.5.1	Docker	19
2.1.5.2	Strumenti	20
3	Processi di supporto	21
3.1	Documentazione	21
3.1.1	Introduzione	21
3.1.1.1	Documentation as Code	21
3.1.2	Ciclo di vita dei documenti	22
3.1.2.1	I redattori	23
3.1.2.2	I verificatori	24
3.1.2.3	Il responsabile	25
3.1.2.4	L'amministratore	25
3.1.3	Struttura del documento	25
3.1.3.1	Verbali: struttura generale	26
3.1.4	Regole tipografiche	27
3.1.5	Abbreviazioni	28
3.1.6	Strumenti	28
3.2	Verifica	29
3.2.1	Introduzione	29
3.2.2	Descrizione	29
3.2.3	Attività	29
3.2.3.1	Verifica dei documenti	29
3.2.3.2	Analisi	31

3.2.3.3	Testing	32
3.2.3.4	Test di unità	33
3.2.3.5	Test di integrazione	33
3.2.3.6	Test di sistema	34
3.2.3.7	Test di regressione	34
3.2.3.8	Test di accettazione	34
3.2.3.9	Codici dei test	35
3.2.3.10	Stato dei test	35
3.2.3.11	Strumenti	35
3.3	Gestione della configurazione	36
3.3.1	Introduzione	36
3.3.2	Numeri di versionamento	36
3.3.3	Tecnologie	36
3.3.4	Repository	37
3.3.4.1	Struttura repository	37
3.3.4.2	Sincronizzazione e Branching	37
3.3.5	Controllo di configurazione	38
3.3.5.1	Change request (Richiesta di modifica)	38
3.3.6	Configuration Status Accounting (Contabilità dello Stato di Configurazione)	39
3.3.7	Release management and delivery	39
3.4	Gestione della qualità	40
3.4.1	Introduzione	40
3.4.2	Attività	40
3.4.3	Strumenti	41
3.4.4	Struttura e identificazione metriche	41
3.4.5	Criteri di accettazione	41
4	Processi organizzativi	42
4.1	Gestione dei Processi	42
4.1.1	Pianificazione	43
4.1.1.1	Assegnazione dei ruoli	43

4.1.1.2	Responsabile	43
4.1.1.3	Amministratore	44
4.1.1.4	Analista	44
4.1.1.5	Progettista	44
4.1.1.6	Programmatore	45
4.1.1.7	Verificatore	45
4.1.1.8	Ticketing	45
4.1.2	Coordinamento	47
4.1.2.1	Comunicazione	47
4.1.2.2	Comunicazioni sincrone	47
4.1.2.3	Comunicazioni asincrone	47
4.1.2.4	Riunioni interne	48
4.1.2.5	Riunioni esterne	49
4.2	Miglioramento	50
4.2.1	Scopo	50
4.2.2	Analisi	50
4.2.3	Miglioramento	50
4.3	Formazione	51
4.3.1	Scopo e aspettative	51
4.3.2	Metodo di formazione	51
4.3.2.1	Individuale	51
4.3.2.2	Di gruppo	51

1 Introduzione

1.1 Finalità del documento

L'obiettivo fondamentale del seguente documento è quello di stabilire le linee guida e le *best practice* che ciascun membro del gruppo deve seguire per garantire un approccio efficiente ed efficace nel processo di realizzazione del progetto didattico.

I processi e le relative attività contenute nel seguente documento sono state definite a partire dallo Standard ISO/IEC 12207:1995.

Il documento è strutturato secondo i processi del ciclo di vita del software e presenta una gerarchia in cui ogni processo si configura come una serie di attività. Ciascuna attività è composta da procedure dotate di obiettivi, scopi e strumenti ben definiti.

In aggiunta, il documento dettaglia le convenzioni relative all'utilizzo dei diversi strumenti adottati durante lo sviluppo del prodotto.

È importante sottolineare che questo documento è in continua evoluzione poiché le norme definite al suo interno vengono regolarmente riesaminate, aggiornate e ottimizzate seguendo un approccio incrementale.

1.2 Glossario

Nella documentazione è incluso il **Glossario**, dove vengono definiti tutti i termini specifici o potenzialmente ambigui presenti nei vari documenti correlati al progetto. La presenza di una nota a pedice con la lettera G accanto a un termine indica che è possibile trovare la sua definizione nel **Glossario**.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- **Standard ISO/IEC 12207:1995**: Standard internazionale che definisce un modello di ciclo di vita del software e in cui sono definite delle linee guida per la gestione dei processi software e le relative attività.
In sintesi, stabilisce una struttura per organizzare le diverse fasi e le attività coinvolte nel ciclo di vita del software, aiutando le organizzazioni a sviluppare prodotti software in modo più efficiente ed efficace.

1.3.2 Riferimenti informativi

- Libro "Clean Code" di Robert C. Martin.

2 Processi primari

2.1 Sviluppo

2.1.1 Introduzione

L'ISO/IEC 12207:1995 stabilisce le linee guida per il processo di sviluppo, il quale include attività cruciali come analisi, progettazione, codifica, integrazione, testing, installazione e accettazione. È fondamentale svolgere queste attività in stretta aderenza alle linee guida e ai requisiti definiti nel contratto con il cliente, garantendo così un'implementazione accurata e conforme alle specifiche richieste.

2.1.2 Analisi dei requisiti

L'analisi dei requisiti è una attività critica nello sviluppo del software, poiché stabilisce le basi per il design, l'implementazione e i test del sistema. Secondo lo standard ISO/IEC 12207:1995, lo scopo dell'analisi dei requisiti è di comprendere e definire in modo esaustivo le esigenze del cliente e del sistema.

L'attività di analisi richiede di rispondere a domande fondamentali come: "Qual è il dominio?", "Qual è la cosa giusta da fare?" e consiste nella comprensione approfondita del dominio e nella definizione chiara di obiettivi, vincoli e requisiti sia tecnici che funzionali

2.1.2.1 Obbiettivi:

- Definire insieme al proponente gli obiettivi del prodotto per rispecchiarne le aspettative, comprendendo l'identificazione, documentazione e validazione dei requisiti funzionali e non funzionali;
- Facilitare la comprensione comune tra gli stakeholder e il team di sviluppo;
- Permettere una stima sulle tempistiche e sui costi;
- Fornire ai progettisti requisiti chiari e di facile comprensione;
- Favorire l'attività di verifica e test fornendo dei riferimenti pratici.

2.1.2.2 Documentazione

È compito degli analisti effettuare l'Analisi dei Requisiti, redigendo un documento con il medesimo nome che deve contenere:

- Introduzione: Scopo del documento stesso;

- Descrizione: Analisi del prodotto
 - Obbiettivi del prodotto;
 - Funzionalità del prodotto;
 - Caratteristiche utente;
 - Tecnologie.
- Casi d'uso: Funzionalità offerte dal sistema dal punto di vista dell'utente
 - Attori: Utenti esterni al sistema;
 - Elenco dei casi d'uso:
 - * Casi d'uso in formato testuale;
 - * Diagrammi dei casi d'uso.
- Requisiti:
 - Requisiti funzionali;
 - Requisiti qualitativi;
 - Requisiti di vincolo.

2.1.2.3 Casi d'uso

Forniscono una descrizione dettagliata delle funzionalità del sistema dal punto di vista degli utenti, identificando come il sistema risponde a determinate azioni o scenari. In breve, i casi d'uso sono strumenti utilizzati nell'analisi dei requisiti per catturare e illustrare in modo chiaro e comprensibile come gli utenti interagiranno con il software e quali saranno i risultati di tali interazioni.

Ogni caso d'uso è costituito da:

1. **Identificativo** nel formato:

UC [Numero caso d'uso] . [Numero sotto caso d'uso] - [Titolo]

(ex. UC6.1 - Visualizzazione posizione sensore).

con:

- **Numero caso d'uso:** ID numerico del caso d'uso;
- **Numero sotto caso d'uso:** ID numerico del sottocaso d'uso, presente esclusivamente in caso di presenza di un sottocaso d'uso.
- **Titolo:** Titolo breve ed esplicativo del caso d'uso.

2. **Attore principale:** Entità esterna che interagisce attivamente con il sistema per soddisfare una sua necessità;
3. **Attore secondario:** Eventuale entità esterna che non interagisce attivamente con il sistema, ma all'interno di un caso d'uso permette al sistema di soddisfare il bisogno dell'attore principale.
4. **Descrizione:** Eventuale descrizione breve della funzionalità;
5. **Scenario principale:** Sequenza di eventi che si verificano quando un attore interagisce con il sistema per raggiungere l'obiettivo del caso d'uso (Postcondizioni);
6. **Estensioni:** Scenari alternativi (se presenti) che, in seguito ad una o più specifiche condizioni, portano il flusso del caso d'uso a non giungere alle Postcondizioni (Opzionale);
7. **Precondizioni:** Stato in cui si deve trovare il sistema affinché la funzionalità sia disponibile all'attore;
8. **Postcondizioni:** Stato in cui si trova il sistema dopo l'esecuzione dello scenario principale;
9. **User story associata:** Breve descrizione di una funzionalità del software, scritta dal punto di vista dell'utente, che fornisce contesto, obiettivi e valore.
Nella forma: "Come [utente] desidero poter [funzionalità] per [valore aggiunto]."

2.1.2.4 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono strumenti grafici che permettono di visualizzare in modo chiaro e intuitivo le funzionalità offerte dal sistema dal punto di vista dell'utente. Inoltre, consentono di identificare e comprendere rapidamente le relazioni e le interazioni tra i vari casi d'uso, offrendo una visione d'insieme delle dinamiche del sistema.

I diagrammi dei casi d'uso si concentrano sulla descrizione delle funzionalità del sistema dal punto di vista degli utenti senza approfondire dettagli implementativi. La loro finalità principale è evidenziare le interazioni esterne al sistema, offrendo una visione focalizzata sulle funzionalità e sull'interazione dell'utente con il sistema stesso. Un diagramma dei casi d'uso fornisce una panoramica visuale delle interazioni chiave tra gli attori e il sistema, facilitando la comprensione dei requisiti funzionali del sistema e la comunicazione tra gli stakeholder del progetto. Di seguito sono elencati i principali componenti di un diagramma dei casi d'uso:

- **Attori:** Gli attori sono le entità esterne al sistema che interagiscono con esso e possono essere utenti umani, altri sistemi software o componenti esterne.

Gli attori sono rappresentati come "stickman" all'esterno del rettangolo che delinea il sistema.

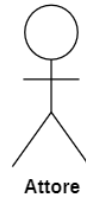


Figure 1: Rappresentazione Attore

- **Casi d'Uso:** I casi d'uso identificano le diverse funzionalità offerte dal sistema con cui l'attore può interagire. Ogni caso d'uso viene rappresentato tramite una forma ovale contenente un ID ed un titolo esplicativo.

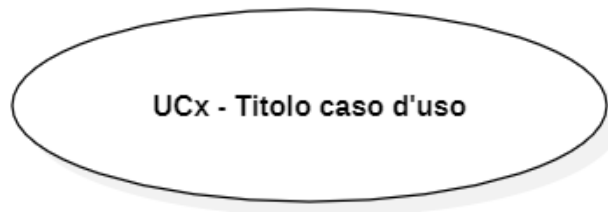


Figure 2: Rappresentazione caso d'uso

- **Sottocasi d'uso:** Un sottocaso d'uso rappresenta una versione più dettagliata di un caso d'uso più generico, offrendo un livello di dettaglio più approfondito sulle funzionalità o sui particolari scenari di utilizzo rispetto al caso d'uso principale.

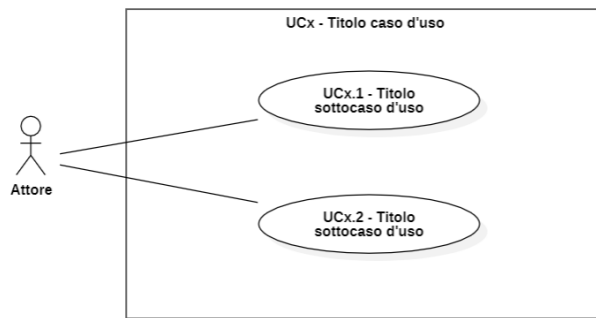


Figure 3: Rappresentazione sottocaso d'uso

- **Sistema:** Il sistema viene rappresentato da un rettangolo e viene identificato con un titolo. All'interno del sistema saranno collocati i casi d'uso, mentre al suo esterno gli attori.

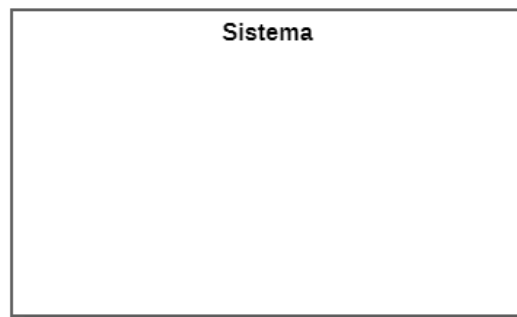


Figure 4: Rappresentazione sistema

- **Relazioni tra Attori e Casi d'Uso:**
 - **Associazione:** Le linee di associazione collegano gli attori ai casi d'uso corrispondenti, indicando quali attori sono coinvolti in una particolare interazione. Più precisamente, una linea di associazione collega un attore a un caso d'uso quando quell'attore è coinvolto nell'interazione descritta dal caso d'uso stesso. Questo legame rappresenta visivamente il ruolo dell'attore nell'utilizzo o nell'avvio di una funzione specifica offerta dal sistema.



Figure 5: Rappresentazione associazione

• **Relazioni tra attori:**

- **Generalizzazione tra attori:** La generalizzazione tra attori rappresenta una relazione di ereditarietà tra attori, dove un attore specializzato (figlio) eredita comportamenti e caratteristiche da un attore base (genitore). Questo aiuta a organizzare gerarchicamente gli attori coinvolti nell'interazione con il sistema nei diagrammi dei casi d'uso.

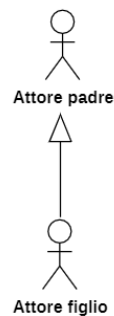


Figure 6: Rappresentazione generalizzazione tra attori

• **Relazioni tra casi d'uso:**

- **Inclusione:** La relazione di inclusione indica che un caso d'uso (detto "includente") include il comportamento di un altro caso d'uso (detto "incluso"). In pratica, quando un attore avvia il caso d'uso includente, il caso d'uso incluso viene eseguito come parte integrante del primo. Questo è utile per evitare la duplicazione del codice o delle azioni in diversi casi d'uso. La relazione di inclusione è spesso rappresentata da una freccia con la punta aperta. Esempio: Supponiamo che ci sia un caso d'uso generico chiamato "Autenticazione". Questo caso d'uso può includere il caso d'uso più specifico "Gestione Password". Quando un utente si autentica, il sistema eseguirà anche la gestione della password come parte del processo di

autenticazione.

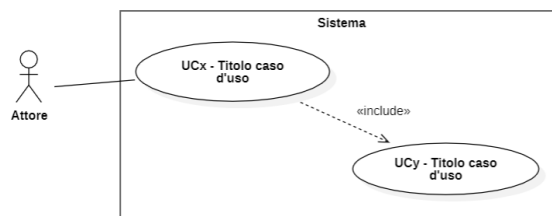


Figure 7: Rappresentazione inclusione

- **Estensione:** La relazione di estensione indica che un caso d'uso (detto "estendente") può estendere il comportamento di un altro caso d'uso (detto "esteso") in determinate circostanze. In altre parole, l'esecuzione del caso d'uso estendente può essere condizionalmente estesa o arricchita dal caso d'uso esteso. La relazione di estensione è spesso rappresentata da una freccia con la punta chiusa. Esempio: Considera un caso d'uso "Gestione Ordini". Questo caso d'uso potrebbe estendere il caso d'uso "Verifica Disponibilità" se durante la gestione degli ordini si verifica la necessità di verificare la disponibilità di un determinato prodotto. In questo modo, l'estensione permette di gestire situazioni specifiche senza ingombrare il flusso principale del caso d'uso base.

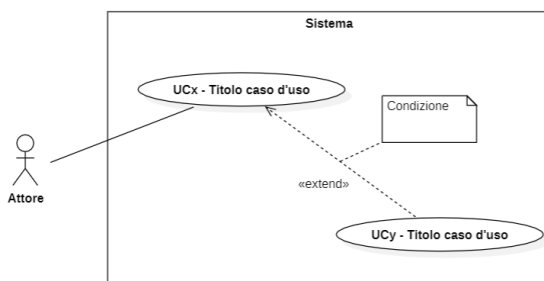


Figure 8: Rappresentazione estensione

- **Generalizzazione casi d'uso:** La generalizzazione nei diagrammi dei casi d'uso rappresenta una relazione di ereditarietà tra casi d'uso, indicando che un caso d'uso più specifico eredita il comportamento da un caso d'uso più generico. Questa relazione è simboleggiata da una linea con una freccia vuota che punta dal caso d'uso più specifico al caso d'uso più generico.

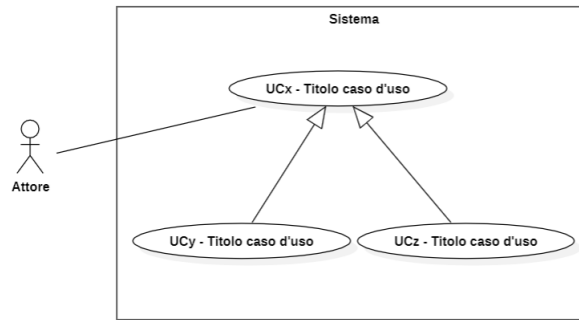


Figure 9: Rappresentazione generalizzazione

2.1.2.5 Requisiti

I requisiti di un prodotto software sono specifiche dettagliate e documentate che delineano le funzionalità, le prestazioni, i vincoli e altri aspetti critici che il software deve soddisfare. Questi requisiti fungono da guida per lo sviluppo, il testing e la valutazione del prodotto, assicurando che risponda alle esigenze degli utenti e agli obiettivi del progetto. Includono

- **Requisiti funzionali:** Descrivono le funzionalità che il software deve avere.
- **Requisiti non funzionali:** Definiscono principalmente criteri di prestazione, qualità, sicurezza e vincoli del sistema, ovvero caratteristiche che non riguardano direttamente le funzionalità specifiche del software.

Una precisa definizione dei requisiti è fondamentale: devono risultare inequivocabili e rispondere pienamente alle attese del cliente o del proponente.

Ogni requisito è costituito da:

1. **Identificativo** nel formato:

R [Abbreviazione tipologia requisito] [Codice]

con:

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

2. **Importanza:**

- **Obbligatorio:** irrinunciabile per qualcuno degli stakeholder;
 - **Desiderabile:** non strettamente necessario ma a valore aggiunto;
 - **Opzionale:** relativamente utile o contrattabile più avanti nel tempo.
3. **Descrizione:** Narrazione chiara e dettagliata che fornisce una spiegazione completa del comportamento o della caratteristica che il software deve possedere;
 4. **Fonte:** Fonte del requisito (ex. capitolato, Verbale interno/esterno);
 5. **Casi d'uso:** Elenco casi d'uso associati.

2.1.2.6 Metriche

Le metriche nell'analisi dei requisiti sono strumenti utilizzati per valutare, misurare e gestire diversi aspetti dei requisiti di un sistema o di un progetto. Queste metriche aiutano a garantire che i requisiti siano completi, corretti, coerenti e comprensibili. Alcune metriche comuni utilizzate nell'analisi dei requisiti includono:

- **Completezza dei requisiti:** Misura la quantità di requisiti identificati rispetto a quelli effettivamente necessari per il sistema. Una metrica comune è la percentuale di requisiti identificati rispetto a quelli totali;
- **Coerenza:** Valuta la coerenza tra i requisiti. Ad esempio, si potrebbe misurare il numero di conflitti o di requisiti duplicati;
- **Tracciabilità:** Indica la capacità di tracciare i requisiti attraverso le fasi del progetto. Si possono usare metriche come il numero di requisiti tracciati rispetto al totale;
- **Comprensibilità:** Misura la chiarezza e la comprensibilità dei requisiti. Si potrebbe valutare tramite sondaggi o questionari la facilità con cui gli stakeholder comprendono i requisiti;
- **Stabilità:** Misura quanto i requisiti cambiano nel tempo. Si possono usare metriche come il tasso di cambiamento dei requisiti per periodo;
- **Priorità:** Valuta l'importanza relativa dei requisiti. Si possono assegnare punteggi di priorità ai requisiti e valutare la distribuzione di questi punteggi;
- **Testabilità:** Misura la facilità con cui i requisiti possono essere verificati tramite test. Si potrebbe valutare la quantità di requisiti che possono essere testati e quelli che richiedono ulteriori specifiche per la verifica;

- **Misurazione del rischio:** Valuta il livello di rischio associato ai requisiti. Si potrebbero utilizzare valutazioni qualitative o quantitative per attribuire livelli di rischio a ciascun requisito.

Ognuna di queste metriche è necessaria ad assicurare che i requisiti siano gestiti in modo efficace e che siano allineati alle esigenze degli stakeholder.

2.1.2.7 Strumenti

2.1.3 Progettazione

2.1.3.1 Scopo

L'attività di progettazione consiste nel processo di definizione e documentazione di:

- Requisiti;
- Specifiche tecniche;
- Architettura del prodotto software e unità architetturali.

al fine di ottenere un prodotto adeguato alle necessità degli stakeholder.

Inoltre la progettazione ha lo scopo di:

- Assicurare la qualità del prodotto finale;
- Suddividere i compiti di implementazione, rendendoli più semplici ed efficienti;
- Ottimizzare i tempi e le risorse necessarie per il completamento del progetto;

La progettazione costituisce un procedimento inverso all'Analisi dei Requisiti; essa si configura come un articolato processo volto a gestire la complessità del prodotto mediante l'organizzazione e la distribuzione delle responsabilità di realizzazione. Si avvale di un approccio sistematico per affrontare e superare le sfide intrinseche, ricorrendo a pattern e best practice al fine di individuare soluzioni congrue alle esigenze delineate nei requisiti.

2.1.3.2 Descrizione

2.1.3.3 Aspettative

2.1.3.4 Strumenti

- **StarUML:** È un'applicazione software utilizzata per creare diagrammi e modelli UML nel processo di sviluppo del software.
Aiuta a visualizzare graficamente i vari aspetti di un sistema, consentendo agli utenti di creare e modificare facilmente diagrammi UML come diagrammi dei casi d'uso, delle classi, di attività e altri, semplificando così il processo di progettazione e analisi dei sistemi software.

2.1.4 Codifica

2.1.4.1 Scopo

L'attività di codifica è affidata al programmatore e rappresenta il momento cruciale in cui le funzionalità richieste dal proponente prendono forma.

Durante questa fase, le idee e i concetti delineati dai progettisti vengono tradotti in codice, creando istruzioni e procedure che i calcolatori possono eseguire.

I programmatori devono rispettare scrupolosamente le linee guida e le norme stabilite per garantire che il codice sia in linea con le specifiche stabilite e che traduca in modo accurato le concezioni iniziali dei progettisti.

2.1.4.2 Aspettative

La codifica è finalizzata alla creazione di un prodotto software in linea con le richieste del committente e conforme agli accordi stipulati.

Il rigoroso rispetto delle norme garantisce la creazione di codice di alta qualità, facilitando la manutenzione, l'estensione e la verifica del software, contribuendo costantemente al miglioramento della sua qualità complessiva.

2.1.4.3 Norme di codifica

Le seguenti norme sono state formalizzate prendendo spunto dal libro "Clean Code" di Robert C. Martin.

Nomi significativi

Usare nomi che riflettano il significato e lo scopo delle variabili, funzioni e classi. Evitare abbreviazioni ambigue.

Indentazioni e formattazione consistente

L'utilizzo di un tab per ciascun livello di annidamento è fondamentale per assicurare una struttura coerente del codice, migliorandone la comprensione e agevolandone la gestione.

Lunghezza dei metodi

La lunghezza ottimale dei metodi può variare a seconda del contesto e delle best practices di programmazione. Tuttavia, in generale, molti esperti consigliano che i metodi siano brevi e focalizzati su una singola responsabilità. Un principio comune è quello espresso da Robert C. Martin nel suo libro "Clean Code", che suggerisce che i metodi dovrebbero essere idealmente lunghi quanto basta per svolgere una singola operazione e non più lungo di quanto si possa visualizzare senza dover scorrere la pagina.

Questo favorisce:

- **Chiarezza;**
- **Manutenibilità;**
- **Comprensibilità;**
- **Testabilità:** Metodi più brevi sono più facili da testare in isolamento, il che favorisce l'implementazione di test di unità efficaci;
- **Conformità ai principi SOLID:** La brevità dei metodi è spesso correlata al principio Single Responsibility Principle (SRP) dei principi SOLID, il che favorisce la costruzione di codice più modulare e coeso.

Lunghezza righe di codice

Mantenere le righe di codice entro i 80-120 caratteri permette una migliore leggibilità del codice su schermi di diverse dimensioni e facilita la visualizzazione di più file affiancati. Inoltre, limitare la lunghezza delle righe può incoraggiare la scrittura di codice più chiaro e modulare.

2.1.4.4 Commenti

Evitare commenti superflui e non necessari: l'obiettivo è scrivere il codice in modo chiaro e autoesplicativo, riducendo la dipendenza da commenti esplicativi.

2.1.5 Configurazione dell'ambiente di esecuzione

2.1.5.1 Docker

La progettazione dei Docker e la scrittura dei Dockerfile sono considerate parte del processo di sviluppo del software. Le regole e le best practice di codifica per i file Docker sono fondamentali per garantire la creazione, la gestione e la distribuzione efficace dei container:

- **Chiarezza e Coerenza:**
 - Utilizzare nomi descrittivi per le immagini e i container.
 - Mantenere una struttura coerente e consistente per assicurare un'organizzazione uniforme all'interno dei file Docker.
- **Versionamento:**
 - Specificare sempre la versione dell'immagine di base (base image) per garantire la riproducibilità.
 - Evitare di utilizzare tag "latest" per immagini di produzione.
- **Minimizzazione degli strati (Layering):**

- Ridurre il numero di istruzioni nell'esecuzione del Dockerfile per minimizzare gli strati dell'immagine.
 - Raggruppare le istruzioni correlate per sfruttare la cache Docker.
- **Sicurezza:**
 - Utilizzare immagini ufficiali o verificate.
 - Evitare di eseguire processi Docker con privilegi elevati quando possibile.
 - Usare ARG per parametrizzare informazioni sensibili.
- **Ottimizzazione delle risorse:**
 - Limitare l'uso di risorse all'interno dei container (CPU, memoria, etc.).
 - Usare immagini leggere e ottimizzate per la produzione.
- **Gestione delle variabili d'ambiente:**
 - Usare variabili d'ambiente per configurazioni dinamiche.
 - Fornire valori predefiniti opportuni per le variabili d'ambiente.
- **Logging e monitoraggio:**
 - Configurare i container per registrare i log in modo efficace.
 - Integrare strumenti di monitoraggio, se necessario.
- **Pulizia e riduzione delle dimensioni:**
 - Pulire i pacchetti temporanei e le risorse non necessarie dopo l'installazione delle dipendenze.
 - Ridurre le dimensioni delle immagini utilizzando multi-stage builds.
- **Documentazione:**
 - Aggiungere commenti nel Dockerfile per spiegare le decisioni di progettazione.
 - Fornire una documentazione chiara su come utilizzare e configurare il container.
- **Testing:**
 - Implementare test automatizzati per il Dockerfile e i container, se possibile.

2.1.5.2 Strumenti

- **VSCode**
- **Docker**
- **Git**

3 Processi di supporto

3.1 Documentazione

3.1.1 Introduzione

La documentazione costituisce l'insieme di informazioni scritte che accompagnano un prodotto software, fornendo dettagli utili a sviluppatori, distributori e utenti.

Tra gli scopi della documentazione troviamo:

- Comprensione del prodotto senza supporto umano, usandola come mezzo di comunicazione;
- Segnare il confine tra creatività e disciplina;
- Assicurare che i processi produttivi si svolgano con la qualità attesa.

L'obiettivo della sezione è:

- Definire delle procedure ripetibili che permettano di uniformare la documentazione prodotta dal gruppo ed il metodo di lavoro;
- Raccogliere ed organizzare le norme che i membri del team devono seguire così da semplificare l'operazione di scrittura dei documenti.

Tali norme dovranno essere applicate da tutti i membri del team, ed i sorgenti \LaTeX che contengono tale documentazione verranno inseriti nel repository disponibile all'indirizzo:

<https://github.com/ByteOps-swe/Sorgente-documenti>

3.1.1.1 Documentation as Code

L'approccio che si intende adottare è quello di "Documentation as Code" (Documentazione come Codice) che consiste nel trattare la documentazione di un progetto software allo stesso modo in cui si tratta il codice sorgente. Questo approccio è incentrato sull'utilizzo di pratiche e strumenti tipici dello sviluppo software per creare, gestire e distribuire la documentazione.

Alcuni aspetti chiave di "Documentation as Code" includono:

- **Versionamento**
- **Scrittura in Formato Testuale**
- **Automazione**
- **Collaborazione**

- **Integrazione Continua**

- **Distribuzione**

Questo approccio porta diversi vantaggi, tra cui una maggiore coerenza, una migliore tracciabilità delle modifiche e facilità di manutenzione. Inoltre, il concetto di "Documentation as Code" si allinea con la filosofia DevOps, dove la collaborazione e l'automazione sono valori chiave.

3.1.2 Ciclo di vita dei documenti

Il ciclo di vita dei documenti è una sequenza di stati e attività:

1. **Stato: Necessità**

- (a) Nasce la necessità di una documentazione, per obbligo o per opportunità;
- (b) Pianificazione della sua stesura;
- (c) Suddivisione in sezioni;
- (d) Durante le riunioni, si procede alla discussione e alla definizione collettiva di una traccia per il contenuto;
- (e) Assegnazione delle sezioni ai redattori tramite task su ITS.

2. **Stato: Redazione**

- (a) Ogni tipo di documento viene creato secondo la struttura specificata nei prossimi paragrafi;
- (b) Il team realizza il documento redigendone il contenuto rispettando le norme definite;

3. **Stato: Verifica**

- (a) Quando completato il documento viene revisionato dai verificatori;
- (b) Il documento viene compilato e il PDF generato viene inserito nella repository all'indirizzo: <https://github.com/ByteOps-swe/Documents> ;

4. **Stato: Manutenzione**

- (a) Manutenzione: ogni documento sotto configuration management deve essere modificato in accordo alle norme di versionamento e di change management.
- (b) La richiesta di modifica nasce da una nuova necessità sul contenuto del documento e riporta il documento allo stato omonimo.

3.1.2.1 I redattori

Il redattore responsabile della redazione di un documento o di una sua sezione deve seguire lo stesso approccio richiesto per la codifica del software, adottando il workflow noto come "feature branch".

Caso redazione nuovo documento, sezione o modifica

Dalla repository "Sorgente documenti" contenente i sorgenti \LaTeX , il redattore dovrà creare un nuovo branch Git in locale e passare ad esso mediante l'utilizzo del comando:

```
git checkout main
git checkout -b branch_identificativoTask
```

Il nome del branch destinato alla redazione o modifica del documento o di una sua sezione deve essere descrittivo al fine di consentire un'identificazione immediata e precisa del documento o della sezione su cui si sta lavorando. Pertanto, relativamente alla redazione dei documenti, deve essere adottata la seguente **convenzione per la nomenclatura dei branch**:

- Il nome del branch deve essere uguale al nome del documento senza estensioni, senza spazi e versione, scritto in camel case. (ex. VerbaleInterno14_11_2023) (Riferimento: 3.1.4);
- Nel caso della redazione di una sezione si pone alla fine "_NomeSezione". (ex. NormeDiProgetto_Documentazione);
- Nel caso di modifica di una sezione si pone alla fine "_ModNomeSezione". (ex. NormeDiProgetto_ModDocumentazione).

A questo punto, dopo aver creato il documento, la sezione o la modifica assegnata e avviato la stesura, affinché gli altri redattori possano continuare il lavoro, è necessario rendere il branch accessibile anche nella repository remota, seguendo i seguenti passaggi:

```
git add .
git commit -m "Descrizione del commit"
git push origin branch_identificativoTask
```

Caso modifica documento in stato di redazione

Qualora il redattore intenda continuare la stesura di un documento (o sezione) iniziato da un altro redattore, sono necessari i comandi:

```
git pull
git checkout branch_identificativoTask
```

Salvataggio e condivisione progressi di task non completate

Alla fine di una sessione di modifiche di un file, nel caso si desideri rendere accessibile ai membri il lavoro non ancora completato e, pertanto, non pronto alla verifica, è necessario:

1. Eseguire il push_G delle modifiche fatte nel branch

```
git add .
git commit -m "Descrizione del commit"
git push origin branch_identificativoDocumento
```

2. Nel caso di problemi con il punto 1:

```
git pull origin branch_identificativoDocumento
```

3. Risolvi i conflitti e ripeti punto 1.

Completamento compito di redazione

Al termine del loro lavoro, i redattori:

1. Segnalano il completamento dell'attività a loro assegnata (essa sia la stesura completa di un documento o di una sua parte) posizionando l'issue relativa nella colonna "Da revisionare" della [DashBoard documentazione](#).
2. Attuano una **pull request**:
 - (a) Aggiornare il registro delle modifiche inserendo i dati richiesti in una nuova riga e incrementando la versione (Il verificatore è definito all'assegnazione della task, presente nella descrizione della Issue dell'ITS).
 - (b) Eseguire i passaggi dettagliati nel caso "**Salvataggio e condivisione progressi di task non completate**".
 - (c) Vai sul repository su GitHub » Apri la sezione "pull request" » Clicca "New pull request"
 - (d) Seleziona come branch di destinazione "main" e come branch sorgente il ramo utilizzato per la redazione del documento (o sezione) da validare
 - (e) Clicca "Create pull request"
 - (f) Dai un titolo e una breve descrizione alla pull request » Seleziona i verificatori su "Reviewers" » Clicca "Create pull request"

Se i verificatori non convalidano il documento (o sezione), i redattori riceveranno feedback allegati alla pull request relativi ai problemi identificati.

3.1.2.2 I verificatori

I compiti e le procedure dei verificatori sono dettagliate al paragrafo 3.2.3.1.

3.1.2.3 Il responsabile

Nel processo di redazione dei documenti, il compito del responsabile consiste nel:

- **Identificare i documenti o le sezioni da redigere.**
- **Stabilire le scadenze entro cui devono essere completati.**
- **Assegnare i redattori e i verificatori ai task.**
- **Approvazione:** Prima della conclusione del suo mandato, il responsabile si riserva il diritto di approvare il lavoro o richiedere eventuali ulteriori modifiche su tutti i documenti redatti e verificati nel periodo in cui ha esercitato la propria carica.

3.1.2.4 L'amministratore

Nel processo di redazione dei documenti, il compito dell'amministratore consiste nel:

- **Inserire nel ITS le attività specificate dal responsabile:**
 - Redattori: assegnatari della issue;
 - Verificatori: specificati nella descrizione della issue;
 - Scadenza.

3.1.3 Struttura del documento

I documenti ufficiali seguono un rigoroso e uniforme schema strutturale, il quale richiede un'osservanza scrupolosa.

Prima pagina

Nella prima pagina di ogni documento deve essere presente:

- Nome e mail del gruppo
- Nome del documento
- Redattori
- Verificatori
- Destinatari

Indice

Tutti i documenti generici devono essere provvisti di un indice dove saranno elencate le varie sezioni e sottosezioni, con la possibilità di raggiungerle direttamente tramite click. In caso di presenza di figure nel documento, sarà presente anche un indice relativo.

Pi  di pagina

In ogni pi  di pagina deve essere presente:

- Nome del gruppo
- Nome del documento
- Numero di pagina

Registro delle modifiche

Tutti i documenti devono essere provvisti di un registro delle modifiche in formato tabellare che contiene un riassunto delle modifiche apportate al documento nel corso del tempo. La tabella deve essere inserita nella sezione registro delle modifiche subito prima dell'indice del documento. La tabella deve registrare le seguenti informazioni:

- **Versione del file.**
- **Data di rilascio.**
- **Autore.**
- **Verificatore.**
- **Descrizione:** un riassunto delle modifiche apportate.

La convenzione per il versionamento   presente al paragrafo : 3.3.2 .

3.1.3.1 Verbali: struttura generale

Questo documento assume l'importante ruolo di costituire un registro ufficiale, atto a riportare in maniera accurata gli argomenti trattati, le decisioni adottate, le azioni da intraprendere e le figure coinvolte.

In particolare,   possibile distinguere tra verbali interni, destinati all'uso interno dell'organizzazione, e verbali esterni, che trovano applicazione quando vi sono terze parti coinvolte nelle discussioni o nelle decisioni documentate.

Nella prima pagina oltre alle informazioni comuni a ogni documento vengono specificate:

- Data della riunione e tipologia (Interna, Esterna) nel nome del documento;
- Luogo: canale di comunicazione adottato;
- Ora di inizio e fine dell'incontro;
- Amministratore;
- Partecipanti della riunione (interni ed esterni);

- Il Responsabile (in basso a destra);

Corpo del documento:

- **Revisione del periodo precedente:** Si valutano gli aspetti positivi, le difficoltà incontrate e si identificano azioni di miglioramento per ottimizzare i processi;
- **Ordine del giorno:** Elenco delle tematiche discusse durante la riunione, accompagnate dai relativi esiti.
Nel caso del verbale esterno, se sono presenti richieste di chiarimenti effettuate alle terze parti coinvolte, saranno incluse nella sottosezione:

- **Richieste e chiarimenti.**

- **Attività da svolgere:** Tabella dove viene specificato:
 - Nome della task da svolgere;
 - Id issue del ITS;
 - Verificatore dell' attività.

Ultima pagina:

- Nel caso di verbale esterno, in ultima pagina, deve essere presente la firma delle terze parti coinvolte, il luogo e la data.

Il template dei verbali è disponibile al link:

<https://github.com/ByteOps-swe/Sorgente-documenti/tree/main/Documents/Verbal/Templates>

3.1.4 Regole tipografiche

Documenti del progetto e nome dei File

Il nome dei documenti generati deve essere omogeneo, con la prima lettera maiuscola ed il resto minuscolo e deve contenere un riferimento alla versione del documento (notazione di versionamento: 3.1.3).

Nello specifico devono seguire la seguente convenzione:

- **Verbali:** Verbale_DD-MM-AAA ;
- **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 .

Regole sintattiche:

- I titoli delle sezioni iniziano con la lettera maiuscola;
- Viene inserito ';' alla fine delle voci dell'elenco tranne l'ultima che termina con '.' . Ogni voce dell'elenco inizia con una lettera maiuscola;
- Le date vengono scritte nel formato GG/MM/AAAA.
- I numeri razionali si scrivono utilizzando la virgola come separatore tra parte intera e parte decimale.

Stile del testo:

- **Grassetto**: Titoli delle sezioni, parole o frasi ritenute di rilievo.
- **Italico**: Riferimento a paragrafi o sezioni, nomi delle aziende e nomi propri dei membri del team.

3.1.5 Abbreviazioni

Nei documenti vi sono molte ripetizioni di termini per la quale si possono usare le seguenti abbreviazioni:

Abbreviazione	Scrittura Estesa
RTB	Requirements and Technology Baseline
PB	Product Baseline
CA	Customer Acceptance
ITS	Issue Tracking System
CI	Configuration Item
SAL	Stato Avanzamento Lavori

3.1.6 Strumenti

Gli strumenti utilizzati dalle attività di redazione dei documenti vogliono soddisfare il principio adottato di "Documentation as Code".

- **GitHub**: versionamento, collaborazione, integrazione continua, automazione e distribuzione;
- **Latex**: scrittura in formato testuale, linguaggio per la stesura di documenti compilati;
- **Overleaf**: per la redazione dei documenti in \LaTeX_G collaborativa;
- **VSCode** : per la redazione con l'utilizzo del plugin \LaTeX Workshop.

3.2 Verifica

3.2.1 Introduzione

Lo scopo dell'attività di verifica nello sviluppo software è garantire la qualità del prodotto attraverso un'analisi accurata e sistematica.

Questo processo coinvolge la revisione e la valutazione delle fasi di sviluppo del codice sorgente e della documentazione, per identificare e correggere errori, difetti o discrepanze rispetto ai requisiti specificati.

La verifica mira a garantire che il software soddisfi gli standard di qualità prestabiliti, riducendo il rischio di errori e migliorando l'affidabilità, la robustezza e l'efficienza del prodotto finale. Questo processo verrà attuato per ciascuna attività e dovrà assicurare che l'output di quest'ultima sia in uno stato stabile, consentendo così l'inizio della successiva fase di validazione.

3.2.2 Descrizione

Il processo di verifica viene eseguito su tutti i processi in esecuzione al raggiungimento di un livello di maturità adeguato o in seguito a modifiche dello stato del processo. Per i processi di supporto, si analizza la qualità dei prodotti generati e dei processi utilizzati al fine di garantire la conformità agli standard di qualità definiti. Le attività relative al processo di verifica vengono assegnate ai verificatori, incaricati di analizzare i prodotti e valutarne la conformità ai vincoli qualitativi specificati nel Piano di Qualifica. Tali operazioni seguono l'ordine specificato nel modello a V. Nel Piano di Qualifica sono documentate tutte le attività che compongono il processo, descrivendone scopi, risultati sperati e risultati ottenuti. Questo fornisce linee guida per una corretta valutazione della qualità, normando e rendendo ripetibile il processo.

3.2.3 Attività

3.2.3.1 Verifica dei documenti

Il ruolo del verificatore nei documenti è cruciale per garantire la qualità e l'accuratezza del contenuto.

Quando il verificatore individua un'Issue nella colonna "Da revisionare" della **DashBoard documentazione**, sarà tenuto a convalidare il file corrispondente presente nella repository "Sorgente documenti".

In aggiunta, il revisore riceverà una notifica via email quando il redattore completa la propria attività, comunicandogli la presenza della pull request assegnatagli.

Nella sezione "pull request" di GitHub, il revisore troverà la richiesta di unire il branch di redazione al branch "main", assumendo il ruolo di revisore. Accedendo alla pull request su

GitHub, il revisore avrà la possibilità di esaminare attentamente il documento in questione e di aggiungere commenti visibili ai redattori nel caso in cui siano necessarie modifiche per la validazione.

Per validare un documento le verifiche da attuare sono:

- **Revisione della correttezza tecnica:** eseguire una revisione tecnica del documento per garantire che tutte le informazioni siano corrette, coerenti e rispettino le norme stabilite;
- **Conformità alle norme:** verificare che il documento segua le linee guida e gli standard prestabiliti per la formattazione, la struttura e lo stile;
- **Consistenza e coerenza:** assicurarsi che il documento sia consistente internamente e coerente con altri documenti correlati. Verificare che non ci siano discrepanze o contraddizioni;
- **Chiarezza e comprensibilità:** valutare la chiarezza del testo, assicurandosi che il linguaggio sia comprensibile per il pubblico di destinazione e che non ci siano ambiguità;
- **Revisione grammaticale e ortografica:** controllare la grammatica, l'ortografia e la punteggiatura del documento per garantire una presentazione professionale;

Dopo l'attuazione dei controlli sopra citati e aver verificato che siano stati adeguatamente rispettati, i passi per convalidare il documento sono i seguenti:

1. **Accetta la Pull Request:** accedere alla pagina della pull request in cui si agirà con il ruolo di revisore nel repository "Sorgente documenti" su GitHub » Risolvere eventuali conflitti » Merge Pull Request.
2. **Elimina il branch:** eliminare il branch creato per la redazione del documento (o sezione).
3. **Sposta la issue in Done:** nella **DashBoard documentazione** spostare la issue relativa al documento validato dalla sezione "Da revisionare" a "Done".
4. **Controllo generazione PDF:** un automazione tramite GitHub Actions compilerà il file \LaTeX e genererà automaticamente il PDF nel branch main della repository **Documents** con la data di redazione nel caso il documento sia un verbale, oppure la versione aggiornata per tutte le altre tipologie di file.

Il verificatore dovrà accedere alla sezione "Actions" di GitHub, supervisionare il processo di compilazione per la possibile generazione di errori, attendere la conclusione del processo di build del codice sorgente e controllare la corretta generazione del documento in formato PDF.

3.2.3.2 Analisi

Attività di controllo su oggetti statici (documentazione, codice) e dinamici (componenti software).

Analisi statica

Analisi condotta sull'oggetto (documentazione e codice) senza eseguirlo. Si prevede l'impiego di metodi di lettura, sia manuali che automatici, al fine di individuare eventuali errori formali. Le due principali tecniche utilizzate sono:

Walkthrough

Il verificatore controllerà nella totalità l'oggetto in cerca di difetti o errori, senza svolgere una ricerca specifica per un certo tipo di errore. Questa metodologia implica appunto un'esaminazione approfondita della documentazione e del codice attraverso un'analisi sistematica. Si promuove la collaborazione tra il verificatore e l'autore del prodotto, strutturata nei seguenti quattro passaggi:

1. **Pianificazione:** una fase di dialogo tra gli autori e i verificatori, finalizzata all'identificazione delle proprietà e dei vincoli che il prodotto deve soddisfare per essere considerato corretto;
2. **Lettura:** il verificatore esamina attentamente i documenti e/o il codice, individuando errori e verificando la conformità ai vincoli prestabiliti;
3. **Discussione:** successivo confronto tra autori e verificatori per discutere l'esito della lettura e valutare eventuali correzioni necessarie;
4. **Correzione:** attività assegnata agli autori per correggere gli errori individuati nei passaggi precedenti.

I walkthrough sono più adatti per attività più creative o nelle fasi iniziali del progetto quando la flessibilità e la discussione sono importanti. Sono ideali quando si desidera una comprensione più approfondita e una valutazione qualitativa.

Le attività in cui verrà applicato Walkthrough sono:

- **Revisione dei requisiti:** per assicurarsi che tutti i requisiti siano chiari, comprensibili e conformi alle esigenze del cliente;
- **Codifica del software:** per identificare errori di logica, pratiche non ottimali o potenziali problemi di manutenibilità nel codice;
- **Documentazione tecnica:** per assicurarsi che la documentazione tecnica sia accurata, chiara e completa.

Inspection

L'obiettivo di questa tecnica consiste nel rilevare eventuali difetti nel prodotto oggetto di analisi mediante una lettura mirata, piuttosto che eseguire una revisione completa del codice e della documentazione allegata.

Tale approccio prevede la specifica preventiva degli elementi da verificare, i quali vengono organizzati in liste di controllo. Queste liste fungono da *checklist* per valutare l'accuratezza dell'attività d'ispezione, determinando se è stata condotta in modo corretto. L'inspection è più adatta quando i documenti o il codice sono complessi e strutturati. È efficace nelle fasi più avanzate del progetto, quando la stabilità e la completezza sono cruciali.

Analisi dinamica

Nel ciclo di sviluppo software, è essenziale condurre una verifica accurata del codice prodotto al fine di garantirne il corretto funzionamento. Questo processo si avvale dell'analisi dinamica, una categoria di metodologie che richiedono l'esecuzione effettiva del prodotto. L'analisi dinamica coinvolge l'esecuzione di una serie di casi di test durante l'attuazione del codice. L'obiettivo di tali test è assicurare l'esecuzione accurata del software e individuare eventuali discordanze tra i risultati ottenuti e quelli previsti.

È importante sottolineare che l'analisi dinamica non è applicabile alla documentazione. Per assicurare l'efficacia dell'analisi dinamica, è necessario automatizzare e rendere ripetibile questo processo, garantendo così una valutazione oggettiva del prodotto. Nel contesto dell'ingegneria del software, il test rappresenta la principale tecnica di analisi dinamica.

3.2.3.3 Testing

Il testing mira a garantire che la componente soggetta al TEST esegua in modo corretto le attività assegnate e aderisca scrupolosamente ai vincoli ad essa attribuiti. Questi test sono inoltre strumentali per individuare eventuali anomalie di funzionamento.

Per ciascun test, è essenziale definire i seguenti parametri:

- **Ambiente:** il sistema (hardware e software) utilizzato per eseguire il test;
- **Stato iniziale:** i parametri del software al momento dell'esecuzione del test;
- **Input:** i dati forniti in input per l'esecuzione del test;
- **Output:** i risultati attesi in output in relazione a uno specifico input;
- **Commenti aggiuntivi:** eventuali osservazioni o annotazioni aggiuntive pertinenti al test;

Tali test andranno poi automatizzati.

3.2.3.4 Test di unità

Questi test sono progettati per verificare singole unità di codice, come funzioni o metodi, in modo isolato e indipendente dal resto del sistema. L'obiettivo principale dei test di unità è garantire che ogni unità di codice funzioni correttamente, conformandosi alle specifiche e restituendo i risultati attesi. Tali test si prestano quindi ad un alto grado di parallelismo, essi vengono pianificati durante la progettazione di dettaglio. Devono essere eseguiti per primi, in quanto verificano l'integrità e la correttezza della singola unità, prima dell'integrazione con le altre.

Per l'implementazione di tali test è concesso utilizzare oggetti simulati o parziali (mocks e stubs) al fine di separare l'unità di codice in esame dalle sue dipendenze esterne, permettendo così la verifica del suo comportamento in contesti controllati, con l'obiettivo di garantire un'isolamento efficace durante le fasi di test.

Un ulteriore obiettivo dei test di unità consiste nel verificare la massima copertura possibile dei percorsi all'interno dell'unità. A tal fine, vengono appositamente progettati per attivare specifici percorsi, creando così una serie di test dedicati che devono garantire una copertura completa del codice dell'unità, generando in tal modo la "structural coverage".

3.2.3.5 Test di integrazione

I test di integrazione sono una fase essenziale nell'analisi dinamica del software e mirano a verificare il corretto funzionamento delle diverse unità di codice quando sono integrate per formare una componente più ampia o l'intero sistema.

Questa fase si concentra sull'interazione tra le parti del software per garantire che lavorino sinergicamente secondo le specifiche del progetto.

I test di integrazione vengono pianificati durante la fase di progettazione architetturale e possono avere un approccio:

- **Top-down:** l'integrazione inizia con le componenti di sistema che presentano maggiori dipendenze e un maggiore valore esterno, consentendo la tempestiva disponibilità delle funzionalità di alto livello.
Ciò consente di effettuare test prolungati sulle funzionalità principali, rendendole disponibili inizialmente ma richiede molti mock;
- **Bottom-up:** l'integrazione ha inizio dalle componenti di sistema caratterizzate da minori dipendenze e un maggiore valore interno, ossia quelle meno visibili all'utente. Questo implica la necessità di meno mock ma ritarda il test delle funzionalità utente esponendole per minor tempo a verifica;

Il team svolgerà, ove possibile, test di integrazione con l'approccio "Top down".

3.2.3.6 Test di sistema

Questi test sono progettati per verificare l'intero sistema software rispetto ai requisiti specificati, garantendo che tutte le componenti siano integrate correttamente e che l'applicazione esegua le funzioni previste in modo accurato e affidabile.

In particolare devono essere implementati:

- **Test End-to-End:** coinvolgono l'esecuzione completa del sistema, dalla sua interfaccia utente fino alle componenti di backend, al fine di simulare l'esperienza completa dell'utente.

I test di sistema vengono eseguiti dopo che sono stati completati con successo i test di integrazione.

3.2.3.7 Test di regressione

Mirano a garantire che le modifiche apportate al codice non abbiano introdotto nuovi difetti o compromesso le funzionalità preesistenti del sistema.

Questi test sono essenziali per assicurare che le modifiche al software non causino regressioni, ovvero la ricomparsa di errori precedentemente risolti o la compromissione di funzionalità precedentemente funzionanti.

I test di regressione devono essere eseguiti ogni volta che vengono apportate modifiche al codice, garantendo una verifica continua e automatica della stabilità del sistema.

3.2.3.8 Test di accettazione

I test di accettazione sono un passo fondamentale prima del rilascio del software e sono progettati per garantire che il prodotto finale sia in grado di soddisfare le aspettative degli utenti finali e che risponda in modo appropriato ai requisiti specificati.

Sequenza delle fasi di test

La sequenza delle fasi di test è la seguente:

1. Test di Unità.
2. Test di Integrazione.
3. Test di Regressione.
4. Test di Sistema.
5. Test di accettazione.

3.2.3.9 Codici dei test

Per classificare ogni test che il team effettuerà durante l'attività di verifica abbiamo deciso di associare un codice identificativo per ciascun test nel formato:

T [tipo] [codice]

dove:

- **[tipo]**: è il tipo di test;
 - **U**: per i test di unità;
 - **I**: per i test di integrazione;
 - **S**: per i test di sistema;
 - **R**: per i test di regressione;
 - **A**: per i test di accettazione;
- **[codice]**: è un numero associato al test all'interno del suo tipo:
 - se il test non ha un padre, è un semplice numero progressivo;
 - se il test ha un padre, sarà nel formato:

[codice.padre] . [codice.figlio]

con:

- * **[codice.padre]** : identifica in maniera univoca il padre del test all'interno della categoria di test relativi al suo tipo;
- * **[codice.figlio]** : numero progressivo per identificare il test;

3.2.3.10 Stato dei test

Ad ogni test verrà successivamente attribuito uno stato che rappresenterà il risultato dell'esecuzione. L'insieme dei risultati dei test sarà registrato nel "cruscotto test".

I test nel cruscotto potranno assumere i seguenti stati:

- **NI**: il test non è ancora stato implementato (Non implementato);
- **S**: il test ha riportato esito positivo (Superato);
- **NS**: il test ha riportato esito negativo (Non Superato);

3.2.3.11 Strumenti

- **Spell checker**: controllo ortografico integrato nell'ambiente di lavoro;
- **Modulo "unittest" Python**.

3.3 Gestione della configurazione

3.3.1 Introduzione

La gestione della configurazione del progetto è una attività che norma il tracciamento e il controllo delle modifiche a documenti e prodotti del software detti Configuration Item (CI). La gestione della configurazione può essere applicata a qualunque categoria di documenti o di "artefatti" che svolga un ruolo nello sviluppo software. Secondo lo standard IEEE, la gestione della configurazione del software è un processo di identificazione, organizzazione e controllo delle modifiche apportate ai prodotti software durante il loro ciclo di vita. Lo standard IEEE 828-2012 definisce la gestione della configurazione del software come "un processo disciplinato per gestire l'evoluzione del software".

3.3.2 Numeri di versionamento

Documentazione:

La convenzione per identificare la versione di un documento è: X.Y.Z con :

- **X:** Viene incrementato al raggiungimento di RTB, PB ed eventualmente CA;
- **Y:** Viene incrementato quando vengono apportate modifiche significative al documento, come cambiamenti strutturali, nuove sezioni importanti o modifiche sostanziali nel contenuto;
- **Z:** Viene incrementato per modifiche minori o aggiornamenti al documento. Questi potrebbero includere correzioni di errori, miglioramenti marginali o l'aggiunta di nuovi contenuti meno rilevanti.

L'incremento dei valori più significati porta i meno significativi a zero.

Ogni variazione di versione deve essere presente nel registro delle modifiche.

3.3.3 Tecnologie

Le tecnologie adottata per la gestione dei configuration item sono:

- **Git:** Version Control System distribuito utilizzato per il versionamento dei CI;
- **GitHub:** Piattaforma web che utilizza Git per il controllo di versione dei CI e per il Ticketing. Utilizzata per la gestione dei change request tramite issue e label e per la contabilità dello stato di configurazione (Branching, posizionamento nella repository e DashBoard di progetto).

3.3.4 Repository

Le repository del team sono:

- **Sorgente-documenti**: Repository per il versionamento, la gestione e lo sviluppo dei sorgenti della documentazione;
- **Documents**: Repository destinata ai committenti/proponenti dove vengono condivisi solo i PDF dei file sorgenti revisionati.
- **proof-of-concept**: Repository destinata al POC.

3.3.4.1 Struttura repository

I repository destinati alla documentazione sono organizzati come segue:

- **Candidatura**: Contenente i documenti richiesti per la candidatura;
 - **Verbali**: Contiene tutti i verbali redatti nel corso del periodo di candidatura, distinti tra verbali esterni e interni;
 - Lettera di presentazione;
 - Valutazione dei costi e assunzione impegni;
 - Valutazione dei capitolati.
- **RTB**: Contenente i documenti richiesti per la revisione omonima;
 - **Verbali**: Contiene tutti i verbali redatti nel corso del periodo della RTB, distinti tra verbali esterni e interni;
 - Piano di Qualifica;
 - Piano di Progetto;
 - Analisi dei Requisiti;
 - Glossario;
 - Norme di Progetto.
- **PB**.

3.3.4.2 Sincronizzazione e Branching

Documentazione: Il processo operativo utilizzato per la redazione della documentazione, noto come "feature branch workflow", implica la creazione di un ramo dedicato per ciascun documento o sezione da elaborare. Tale metodologia permette una parallelizzazione agile dei lavori evitando sovrascritture indesiderate di altri lavori e permette l'adozione dei principi "Documentation as code".

3.3.5 Controllo di configurazione

3.3.5.1 Change request (Richiesta di modifica)

Seguendo lo standard ISO/IEC 12207 per affrontare questo processo in modo strutturato le attività sono:

1. **Identificazione e registrazione:** Le change request vengono identificate, registrate e documentate accuratamente. Questo include informazioni come la natura della modifica richiesta, l'urgenza, l'impatto sul sistema esistente
L'identificazione avviene tramite la creazione di un issue nell'ITS con label: "Change request";
2. **Valutazione e analisi:** Le change request vengono valutate dal team per determinare la loro fattibilità, importanza e impatto sul progetto. Si analizzano i costi e i benefici associati all'implementazione della modifica;
3. **Approvazione o rifiuto:** Il responsabile valuta le informazioni raccolte e decide se approvare o respingere la change request. Questa decisione può essere basata su criteri come il budget, il tempo, le priorità degli stakeholder;
4. **Pianificazione delle modifiche:** Se una change request viene approvata, viene pianificata e integrata nel ciclo di sviluppo del software. Questo può richiedere la rinegoziazione dei tempi di consegna, la revisione del piano di progetto, ecc;
5. **Implementazione delle modifiche:** Le modifiche vengono effettivamente implementate. Durante questo processo, è fondamentale mantenere una traccia accurata di ciò che viene fatto per consentire una corretta documentazione e, se necessario, la possibilità di un rollback;
6. **Verifica e validazione:** Le modifiche apportate vengono verificate per assicurarsi che abbiano raggiunto gli obiettivi previsti e non abbiano introdotto nuovi problemi o errori;
7. **Documentazione:** Tutti i passaggi del processo di gestione delle change request vengono documentati accuratamente per garantire la trasparenza e la tracciabilità. Questa documentazione è utile per futuri riferimenti e per l'apprendimento dalle modifiche apportate;
8. **Comunicazione agli interessati:** Durante tutto il processo, è importante comunicare in modo chiaro e tempestivo agli interessati, come il team di sviluppo, i clienti e altri stakeholder, per mantenere la trasparenza e la fiducia.

3.3.6 Configuration Status Accounting (Contabilità dello Stato di Configurazione)

Questo processo si occupa di registrare e tenere traccia dello stato di tutte le configurazioni di un sistema software durante il suo ciclo di vita.

- **Registrazione delle configurazioni:** Registrazione delle informazioni dettagliate su ogni elemento di configurazione;
 - **Documentazione:** Le informazioni del documento sono presenti nella prima pagina di ciascuno.
- **Stato e cambiamenti:** Tenere traccia dello stato attuale di ciascun elemento di configurazione e di tutti i cambiamenti che avvengono nel corso del tempo. Ciò include le versioni attuali, le revisioni, le modifiche e le baselines;
 - **Registro delle modifiche:** Per monitorare lo stato di ciascun CI, si utilizza il registro delle modifiche incorporato in ognuno di essi;
 - **Branching & DashBoard:** Per visionare la presenza di lavori in atto su di un CI si utilizza a supporto la presenza di branch attivi che lo riguardano e il posizionamento di issue associate nella Dashboard di progetto.
 - * Ogni issue è associato ad un CI tramite label.
- **Supporto per la gestione delle change request:** Registra e documenta le modifiche apportate agli elementi di configurazione in risposta alle richieste di modifica.
 - **ITS:** Per supportare i change request viene utilizzato l'ITS di github, l'identificazione avviene tramite la creazione di un issue nell'ITS con label: "Change request".

La Contabilità dello Stato di Configurazione è un processo fondamentale per mantenere la trasparenza e la tracciabilità nel ciclo di vita del software, aiutando a gestire le configurazioni in modo coerente e a mantenere un registro accurato di tutte le attività e le modifiche che coinvolgono gli elementi di configurazione.

Ad esempio, l'aggiunta di un elemento di configurazione al ramo principale (main) lo designa come la versione più recente e revisionata, mentre le versioni (X.O.O) con $X \geq 1$ indicano l'appartenenza del suddetto elemento di configurazione alla baseline, con la medesima versione.

3.3.7 Release management and delivery

Dopo aver portato a termine le attività nel proprio branch, il responsabile del suo sviluppo è tenuto a avviare una richiesta di pull per incorporare le modifiche nel ramo principale. La richiesta di pull può essere accettata solo se la verifica ha esito positivo.

3.4 Gestione della qualità

3.4.1 Introduzione

Questa sezione è dedicata alle direttive del team per gestire la qualità. Il processo mira a garantire la qualità del flusso operativo adottato dal fornitore e dei prodotti sviluppati, al fine di soddisfare le aspettative del cliente e del proponente.

La gestione della qualità costituisce un approccio olistico che abbraccia l'intero ciclo di vita del software, dall'atto del concepimento all'implementazione e oltre, al fine di assicurare che il prodotto finale sia conforme agli standard di qualità prestabiliti e consenta un miglioramento costante dei processi.

3.4.2 Attività

Le attività che il team si impegna a svolgere per assicurare qualità dei processi e di conseguenza dei prodotti sono:

1. **Definizione degli Standard di qualità:** La gestione della qualità inizia con la definizione chiara degli standard di qualità che il software dovrebbe raggiungere. Questi standard possono includere requisiti funzionali e non funzionali;
2. **Pianificazione della qualità:** Viene sviluppato un piano di qualità che identifica attività specifiche, risorse e tempistiche per garantire la qualità del software durante l'intero ciclo di vita del progetto;
3. **Assicurazione della qualità:** La fase di assicurazione della qualità coinvolge attività continue di monitoraggio e valutazione per garantire che i processi di sviluppo siano conformi agli standard di qualità stabiliti;
4. **Controllo della qualità:** Il controllo della qualità include l'esecuzione di test e verifiche per garantire che il software soddisfi gli standard di qualità e risponda alle specifiche richieste;
5. **Gestione delle modifiche:** Un sistema di gestione delle modifiche è implementato per gestire e controllare le modifiche al software. Questo assicura che ogni modifica venga valutata in termini di impatto sulla qualità complessiva del prodotto;
6. **Miglioramento continuo e correzione:** La gestione della qualità promuove il miglioramento continuo dei processi. Attraverso la raccolta di feedback, l'analisi delle prestazioni e l'implementazione di best practice, il team cerca costantemente di ottimizzare la qualità del software;

7. **Coinvolgimento degli stakeholder:** Gli stakeholder, inclusi clienti e utenti finali, sono coinvolti nel processo di gestione della qualità. I loro feedback sono preziosi per garantire che il software risponda alle esigenze e alle aspettative;
8. **Formazione e competenza del team:** La formazione continua del team è essenziale per mantenere elevate competenze e conoscenze. Un team ben addestrato è in grado di produrre software di alta qualità.

3.4.3 Strumenti

Gli strumenti impiegati per la gestione della qualità sono rappresentati dalle metriche.

3.4.4 Struttura e identificazione metriche

- **Codice:** Identificativo della metrica nel formato:

M [numero] [abbreviazione]

dove:

- M: sta per metrica;
 - [numero]: è un numero progressivo univoco per ogni metrica;
 - [abbreviazione]: è una abbreviazione composta dalle iniziali del nome della metrica.
- **Nome:** Specifica il nome della metrica;
 - **Descrizione:** Breve descrizione della funzionalità della metrica adottata;
 - **Scopo:** Il motivo per cui è importante tale misura al fine del progetto;

Eventualmente anche:

- **Formula:** Come viene calcolata;
- **Strumento:** Lo strumento che viene usato per calcolarla;

3.4.5 Criteri di accettazione

Per ciascuna metrica nel documento "Piano di Qualifica" vengono definiti in formato tabellare:

- **Valore accettabile:** Valore considerato accettabile una volta assunto dalla metrica;
- **Valore preferibile:** Valore ideale che dovrebbe essere assunto dalla metrica;

4 Processi organizzativi

Lo sviluppo software è un processo complesso e multidisciplinare che richiede una pianificazione accurata, una gestione efficiente delle risorse e un controllo rigoroso della qualità. L'adozione di processi organizzativi ben strutturati diventa quindi cruciale per garantire il successo di un progetto software.

4.1 Gestione dei Processi

La Gestione dei Processi si occupa di stabilire, implementare e migliorare i processi che guidano la realizzazione del software, al fine di raggiungere gli obiettivi prefissati e soddisfare le esigenze degli stakeholder.

Le attività di gestione di processo sono:

1. Definizione dei Processi:

- Identificare e documentare i processi chiave coinvolti nello sviluppo software;
- Stabilire linee guida e procedure per l'esecuzione di ciascun processo.

2. Pianificazione e Monitoraggio:

- Elaborare piani dettagliati per l'esecuzione dei processi;
- Monitorare costantemente l'avanzamento, l'efficacia e la conformità ai requisiti pianificati;
- Stimare i tempi, le risorse ed i costi

3. Valutazione e Miglioramento Continuo:

- Condurre valutazioni periodiche dei processi per identificare aree di miglioramento;
- Implementare azioni correttive e preventive per ottimizzare i processi.

4. Formazione e Competenze:

- Assicurare che il personale coinvolto nei processi sia adeguatamente formato;
- Mantenere e sviluppare le competenze necessarie per l'efficace gestione dei processi.

5. Gestione dei Rischi:

- Identificare e valutare i rischi associati ai processi;
- Definire strategie per mitigare o gestire i rischi identificati.

4.1.1 Pianificazione

L'attività di pianificazione nello sviluppo software, nell'ambito della gestione dei processi, è un processo fondamentale finalizzato a definire un piano organizzato e coerente per il corretto svolgimento delle attività di sviluppo del software. Tale processo è disciplinato e guidato dal responsabile, il quale ha il compito di predisporre le attività relative alla pianificazione.

In dettaglio, il responsabile verifica la fattibilità del piano organizzato, garantendo che sia eseguibile in maniera corretta e efficiente da parte dei membri del team. I piani associati all'esecuzione del processo devono includere descrizioni dettagliate delle attività e delle risorse necessarie, comprese le tempistiche, le tecnologie impiegate, le infrastrutture coinvolte e il personale assegnato.

L'obiettivo primario della pianificazione è assicurare che ciascun membro del team assuma ogni ruolo almeno una volta durante lo svolgimento del progetto, promuovendo così una distribuzione equa delle responsabilità e un arricchimento delle competenze all'interno del team.

La pianificazione, stilata dal responsabile, è integrata nel documento del **Piano di Progetto**. Questo documento fornisce una descrizione completa delle attività e dei compiti necessari per raggiungere gli obiettivi prefissati in ogni periodo del progetto.

4.1.1.1 Assegnazione dei ruoli

Durante l'intero periodo del progetto, i membri del gruppo assumeranno sei ruoli distinti, ovvero assumeranno le responsabilità e svolgeranno le mansioni tipiche dei professionisti nel campo dello sviluppo software.

I ruoli a disposizione sono:

4.1.1.2 Responsabile

Figura fondamentale che coordina il gruppo, fungendo da punto di riferimento per il committente e il fornitore e svolgendo il ruolo di mediatore tra le due parti.

In particolare si occupa di:

- Gestire le relazioni con l'esterno;
- Pianificare le attività: quali svolgere, data di inizio e fine, assegnazione delle priorità...
- Valutare i rischi delle scelte da effettuare;
- Controllare i progressi del progetto;
- Gestire le risorse umane;
- Approvazione della documentazione;

4.1.1.3 Amministratore

Questa figura professionale è incaricata del controllo e dell'amministrazione dell'ambiente di lavoro utilizzato dal gruppo ed è anche il punto di riferimento per quanto concerne le norme di progetto. Le sue mansioni principali sono:

- Affrontare e risolvere le problematiche associate alla gestione dei processi;
- Gestire versionamento della documentazione;
- Gestire la configurazione del prodotto;
- Redigere ed attuare le norme e le procedure per la gestione della qualità;
- Amministrare le infrastrutture e i servizi per i processi di supporto;

4.1.1.4 Analista

Figura professionale con competenze avanzate riguardo l'attività di analisi dei requisiti ed il dominio applicativo del problema. Il suo ruolo cruciale è quello di identificare, documentare e comprendere a fondo le esigenze e le specifiche del progetto, traducendole in requisiti chiari e dettagliati. Si occupa di:

- Analizzare il contesto di riferimento, definire il problema in esame e stabilire gli obiettivi da raggiungere;
- Comprendere il problema e definire la complessità e i requisiti;
- Redigere il documento *Analisi dei Requisiti*;
- Studiare i bisogni espliciti ed impliciti;

4.1.1.5 Progettista

Il *Progettista* è la figura di riferimento per quanto riguarda le scelte progettuali partendo dal lavoro dell'analista. Spetta al progettista assumere decisioni di natura tecnica e tecnologica, oltre a supervisionare il processo di sviluppo. Tuttavia, non è responsabile della manutenzione del prodotto. In particolare si occupa di:

- Progettare l'architettura del prodotto secondo specifiche tecniche dettagliate;
- Prendere decisioni per sviluppare soluzioni che soddisfino i criteri di affidabilità, efficienza, sostenibilità e conformità ai Requisiti;
- Redige la *Specifica Architeturale* e la parte pragmatica del *Piano di Qualifica*;

4.1.1.6 Programmatore

Il programmatore è la figura professionale responsabile della codifica del software. Il suo ruolo principale consiste nell'implementare codice informatico basato sulle specifiche fornite dall'analista e sull'architettura fornita dal progettista. In particolare, il *Programmatore*:

- Scrivere codice informatico mantenibile in conformità con le *Specifiche di Progetto*;
- Codificare le varie parti dell'architettura elaborata seguendo l'architettura ideata dal Progettista;
- Realizza gli strumenti per verificare e validare il codice;
- Redigere il *Manuale Utente*;

4.1.1.7 Verificatore

La principale responsabilità del *Verificatore* consiste nell'ispezionare il lavoro svolto da altri membri del team per assicurare la qualità e la conformità alle attese prefissate. Stabilisce se il lavoro è stato svolto correttamente sulla base delle proprie competenze tecniche, esperienza e conoscenza delle norme. Si occupa di:

- Verificare che i prodotti siano conformi alle *Norme di Progetto*;
- Verificare la conformità dei prodotti ai requisiti funzionali e di qualità;
- Ricercare ed in caso segnalare eventuali errori;
- Redigere la sezione retrospettiva del *Piano di Qualifica*, descrivendo le verifiche e le prove effettuate durante il processo di sviluppo del prodotto;

4.1.1.8 Ticketing

3.3.6 GitHub è adottato come sistema di tracciamento degli issue (ITS), garantendo così una gestione agevole e trasparente dei compiti da svolgere. L'amministratore ha la facoltà di creare e assegnare specifici compiti identificati dal responsabile, assicurando chiarezza sulle responsabilità di ciascun individuo e stabilendo tempi definiti. Inoltre, ogni membro del gruppo può monitorare i progressi compiuti nel periodo corrente, consultando lo stato di avanzamento dei vari task attraverso le Dashboard:

- **DashBoard**: Per una chiara visione dello stato dei task;
- **RoadMap**: Per visione delle scadenze dei task.

La procedura da seguire in caso di necessità di svolgere un compito è la seguente:

1. L' amministratore crea e assegna il task su GitHub, seguendo la convenzione descritta in seguito;
2. L'incaricato apre una branch su GitHub seguendo la denominazione suggerita;
3. All'inizio del lavoro di produzione il task viene marcato in corso sulla DashBoard GitHub;
4. Finito il lavoro di produzione viene aperta la pull request su GitHub assegnando il verificatore.
5. Il task viene marcato nella colonna "Da revisionare" sulla DashBoard GitHub.
6. Il verificatore si accerta e agisce secondo quanto esposto al punto 3.2 nel caso di appartenenza;
7. Il task viene marcato nella colonna "Done" della DashBoard GitHub.

I task sono creati dall'amministratore e hanno i seguenti attributi:

- **Titolo:** un titolo conciso e descrittivo;
- **Descrizione:**
 - Descrizione testuale di "to-do";
 - Come ultima riga il Verificatore della task. (Verificatore: Mario Rossi.)
- **Assegnatario:** incaricato svolgimento della task;
- **Scadenza:** Data entro la quale la task deve essere completata;
- **labels:** Tag per identificare la categoria della task. (ex. Verbale, Documents, Develop, Bug).
Inoltre per associare ad ogni Issue un CI vengono utilizzati i seguenti label:
 - **NdP:** Norme di progetto;
 - **PdQ:** Piano di Qualifica;
 - **PdP:** Piano di Progetto;
 - **AdR:** Analisi dei Requisiti;
 - **Poc:** Proof of concept;
 - **Gls:** Glossario.
- **Milestone:** Milestone associata alla task.

4.1.2 Coordinamento

Il coordinamento rappresenta l'attività che sovrintende la gestione della comunicazione e la pianificazione degli incontri tra le diverse parti coinvolte in un progetto di ingegneria del software. Questo comprende sia la gestione della comunicazione interna tra i membri del team del progetto, sia la comunicazione esterna con il proponente e i committenti. Il coordinamento risulta essere cruciale per assicurare che il progetto proceda in modo efficiente e che tutte le parti coinvolte siano informate e partecipino attivamente in ogni fase del progetto.

4.1.2.1 Comunicazione

Il gruppo *ByteOps* mantiene comunicazioni attive, sia interne che esterne al team, le quali possono essere sincrone o asincrone a seconda delle necessità.

4.1.2.2 Comunicazioni sincrone

Comunicazioni sincrone interne

Per le comunicazioni sincrone interne, il gruppo *ByteOps*, ha scelto di adottare *Discord_G* in quanto permette di comunicare tramite chiamate vocali, videochiamate, messaggi di testo, media e file in chat private o come membri di un "*server Discord*".

Comunicazioni sincrone esterne

Per le comunicazioni sincrone esterne, in accordo con l'azienda proponente si è deciso di utilizzare *google meet*.

4.1.2.3 Comunicazioni asincrone

Comunicazioni asincrone interne

Le comunicazioni asincrone interne al nostro team avvengono tramite l'applicazione *Telegram_G* all'interno di un Gruppo dedicato, il quale consente una comunicazione rapida tra tutti i membri del gruppo. Inoltre, tramite la stessa piattaforma, è possibile avere conversazioni dirette e private (*chat*) tra due membri.

Comunicazioni asincrone esterne

Per le comunicazioni asincrone esterne sono stati adottati due canali differenti:

- **E-mail** La posta elettronica è stata utilizzata per comunicare con il committente e, nelle prime fasi del progetto, per coordinare gli incontri con l'azienda proponente. Questa forma di comunicazione offre la flessibilità necessaria per coordinare incontri sincroni e revisioni in modo efficace, consentendo a entrambe le parti di pianificare le interazioni in

base alla propria disponibilità.

- **Element** E' un client di messaggistica istantanea libero ed open source che supporta conversazioni strutturate e crittografate. La sua flessibilità nell'adattarsi a varie esigenze di comunicazione, inclusa la possibilità di condividere file, immagini e altri documenti, ha reso la piattaforma un'opzione versatile e completa per soddisfare le esigenze specifiche del nostro contesto lavorativo.

Riunioni

In ogni riunione, qualunque ne sia la tipologia, verrà designato un segretario con l'incarico di prendere appunti durante il meeting e successivamente redigere un verbale completo, documentando gli argomenti trattati e i risultati emersi durante le discussioni

4.1.2.4 Riunioni interne

Il nostro team ha adottato un approccio settimanale attraverso incontri simili a "stand-up meetings" al fine di facilitare una comunicazione costante e coordinare il progresso delle attività interne.

Le riunioni sono fissate ogni venerdì alle ore:

- **10:00**, nel caso in cui non sia prevista un SAL con l'azienda proponente nello stesso giorno;
- **11:30**, nel caso in cui sia prevista un SAL con l'azienda proponente nello stesso giorno, verranno quindi anche verbalizzate le nuove attività sorte dall'incontro con la proponente.

In caso risultasse necessario, ogni membro del gruppo può decidere di richiedere una riunione supplementare. In questo caso la data e l'orario verranno stabilite tramite il canale Telegram dedicato creando un sondaggio.

Questi incontri rivestono un ruolo cruciale nel monitorare il progresso delle mansioni assegnate, valutare i risultati conseguiti e affrontare le sfide che possono sorgere. Durante questi momenti, i membri del team condividono gli aggiornamenti sulle proprie attività, identificano le problematiche riscontrate e discutono di opportunità di miglioramento nei processi di lavoro. Questo ambiente aperto e collaborativo favorisce l'innovazione e la condivisione di nuove prospettive.

Per agevolare la comunicazione sincrona, il canale utilizzato per questi incontri è *Discord_G*, ritenuto particolarmente efficace per tali scopi.

Sarà compito del responsabile:

- Stabilire preventivamente i punti dell'ordine, considerando l'aggiunta di nuovi durante il corso della riunione;

- Guidare la discussione e raccogliere i pareri dei membri in maniera ordinata;
- Nominare un segretario per la riunione;
- Pianificare e proporre le nuove attività da svolgere.

Verbali interni

Lo svolgimento di una riunione ha come obiettivo la risoluzione dei punti stilati nell'ordine del giorno, la pianificazione delle nuove attività e la retrospettiva del periodo precedente. Al termine di ogni riunione viene creato un issue sull'ITS di GitHub che prevede la stesura del verbale interni. Sarà cura del segretario redigere il verbale includendo tutte le informazioni di rilievo sorte durante la riunione. Le linee guida per la redazione dei verbali interni sono reperibili alla sezione 3.1.3.1

4.1.2.5 Riunioni esterne

Durante il corso del progetto, si renderà necessaria l'organizzazione di vari incontri con i *Committenti* e/o il *Proponente* allo scopo di valutare lo stato di avanzamento del prodotto. La convocazione di tali incontri è di competenza del *Responsabile*, il quale è incaricato di pianificarli e di agevolarne lo svolgimento in maniera efficiente. Sarà compito del Responsabile anche l'esposizione dei punti di discussione al proponente/committente, lasciando la parola ai membri del gruppo interessati quando necessario.

Questo approccio assicura una comunicazione efficace tra il nostro team e i rappresentanti aziendali, garantendo una gestione ottimale del tempo e una registrazione accurata delle informazioni rilevanti emerse durante gli incontri.

I membri del gruppo si impegnano a garantire la propria presenza in modo costante alle riunioni, facendo il possibile per riorganizzare eventuali altri impegni al fine di partecipare. Nel caso in cui gli obblighi inderogabili di un membro del gruppo rendessero impossibile la partecipazione, il responsabile assicurerà di informare tempestivamente il proponente o i committenti, richiedendo la possibilità di rinviare la riunione ad una data successiva.

Riunioni con la proponente

In accordo con l'azienda proponente si è deciso di attuare con cadenza bisettimanale gli incontri di stato avanzamento lavori (SAL) tramite google meet.

Durante tali incontri, si affrontano diversi aspetti, tra cui:

- Discussione delle attività svolte nel periodo precedente, valutando l'aderenza alle concordanze stabilite e identificando eventuali problematiche riscontrate.
- Pianificazione delle attività per il prossimo periodo, definendo gli obiettivi e le azioni necessarie per il loro raggiungimento;

- Chiarezza e risoluzione di eventuali dubbi emersi nel corso delle attività svolte.

Verbali esterni

Come per il caso delle riunioni interne verrà redatto un Verbale con le stesse modalità descritte in precedenza. Le linee guida per la redazione dei verbali esterni sono reperibili alla sezione 3.1.3.1

La firma di approvazione dell'esterno è necessaria per ogni verbale esterno in ultima pagina.

4.2 Miglioramento

4.2.1 Scopo

Secondo lo standard ISO/IEC 12297:1995, il processo di miglioramento nel ciclo di vita del software è finalizzato a stabilire, misurare, controllare e migliorare i processi che lo compongono. L'attività di miglioramento è composta da:

- **Analisi:** Identificare le aree di miglioramento dei processi;
- **Miglioramento:** Implementare le modifiche necessarie per migliorare i processi. di sviluppo del software;

4.2.2 Analisi

Esame Questa operazione richiede di essere eseguita regolarmente e ad intervalli di tempo appropriati e costanti. L'analisi fornisce un ritorno sulla reale efficacia e correttezza dei processi implementati, permettendo di identificare prontamente quelli che necessitano di miglioramenti.

Durante ogni riunione, il team dedica inizialmente del tempo per condurre una retrospettiva sulle attività svolte nell'ultimo periodo. Questa pratica implica una riflessione approfondita su ciò che è stato realizzato, coinvolgendo tutti i membri nella identificazione delle aree di successo e di possibili miglioramenti. L'obiettivo principale è formulare azioni correttive da implementare nel prossimo sprint, promuovendo così un costante feedback e un adattamento continuo per migliorare le prestazioni complessive del team nel corso del tempo.

4.2.3 Miglioramento

Il team implementa le azioni correttive stabilite durante la retrospettiva, successivamente valuta la loro efficacia e le sottopone nuovamente a esame durante la retrospettiva successiva.

L'esito di ogni azione correttiva sarà documentato nella sezione "Revisione del periodo precedente" di ogni verbale.

4.3 Formazione

4.3.1 Scopo e aspettative

L'obiettivo di questa iniziativa è stabilire standard per il processo di apprendimento all'interno del team, assicurando la comprensione adeguata delle conoscenze necessarie per la realizzazione del progetto.

Si prevede che il processo di formazione del Team assicuri che ciascun membro acquisisca una competenza adeguata per utilizzare consapevolmente le tecnologie selezionate dal gruppo per la realizzazione del progetto.

4.3.2 Metodo di formazione

4.3.2.1 Individuale

Ciascun membro del team si impegnerà in un processo di autoformazione per adempiere alle attività assegnate al proprio ruolo. Durante la rotazione dei ruoli, ogni membro del gruppo condurrà una riunione con il successivo occupante del suo attuale ruolo, trasmettendo le conoscenze necessarie. Al contempo, terrà una riunione con chi ha precedentemente svolto il ruolo che esso assumerà, con l'obiettivo di apprendere le competenze richieste.

4.3.2.2 Di gruppo

Sono programmate sessioni formative, condotte dalla proponente, al fine di trasferire competenze relative alle tecnologie impiegate nel contesto del progetto. La partecipazione del team a tali riunioni è obbligatoria.