QB SOFTWARE

×

UNIVERSITÀ DEGLI STUDI DI PADOVA

CORSO DI INGEGNERIA DEL SOFTWARE

ANNO ACCADEMICO 2023/2024

Norme di Progetto

Contatti: qbsoftware.swe@gmail.com

Registro delle modifiche

V.	Data	Membro	Ruolo	Descrizione
2.0.0	05/05/2024	A. Domuta	Responsabile	Approvazione documento
1.2.0	12/04/2024	A. Domuta	Verificatore	Controllo qualità
	12/04/2024	A. Giurisato	Responsabile	Revisione documento
1.1.0	8/03/2024	A. Bustreo	Verificatore	Controllo qualità
	7/03/2024	A. Feltrin	Amministratore	Espansione sezione di progettazione, espansione metriche
1.0.0	25/02/2024	S. Destro	Responsabile	Approvazione documento
0.9.0	24/02/2024	A. Bustreo	Verificatore	Controllo qualità
	23/02/2024	S. Rovea	Amministratore	Revisione documento
0.8.0	17/02/2024	R. Fontana	Verificatore	Controllo qualità
	16/02/2024	S. Rovea	Amministratore	Aggiunta sezione metriche (§B)
0.7.0	16/02/2024	A.Domuta	Verificatore	Controllo qualità
	15/02/2024	S. Destro	Responsabile	Aggiunti miglioramento (§4.2) e formazione (§4.3)
0.6.0	04/02/2024	S. Rovea	Verificatore	Controllo qualità
	02/02/2024	A. Feltrin	Responsabile	Aggiunti processi di verifica (§3.4) e validazione (§3.5)
0.5.0	14/01/2024	S. Rovea	Verificatore	Controllo qualità

V.	Data	Membro	Ruolo	Descrizione
	13/01/2024	R. Fontana	Amministratore	Espansione processi di sviluppo §2.2 e aggiunti standard per la qualità §A
0.4.0	15/12/2023	A. Feltrin	Verificatore	Controllo qualità
	11/12/2023	S. Rovea	Amministratore	Espansione sezione §4.1, aggiunti processi di gestione della configurazione (§3.2) e controllo della qualità (§3.3)
0.3.0	12/12/2023	S. Rovea	Verificatore	Controllo qualità
	10/12/2023	A. Feltrin	Responsabile	Aggiunti processi primari, sezione §2.1 e §2.2
0.2.0	30/11/2023	A. Feltrin	Verificatore	Controllo qualità
	30/11/2023	S. Destro	Amministratore	Aggiunti processi organizzativi, sezione §4.1
0.1.0	19/11/2023	A. Giurisato	Verificatore	Controllo qualità
	18/11/2023	A. Bustreo	Amministratore	Aggiunto processo supporto per la documentazione, sezione §3.1. Aggiunta prefazione, sezione §1

Indice

1 Introduzione

1.1 Scopo e struttura

Con questo documento QB Software intende normare i propri processi per lo sviluppo di un progetto software. Tali processi sono stati creati a partire dallo standard ISO/IEC 12207:1997 proposto durante il corso d'Ingegneria del Software.

Il documento è strutturato nel seguente modo: ogni gruppo di processi è indicato da un numero (a)[1], ogni singolo processo è indicato dal numero (a.b), e ogni attività è indicata dalla numerazione (a.b.c). La scelta di avere un documento per struttura simile allo standard ci permette di mantenere il più fedelmente possibile le linee guida dettate dallo standard stesso. Le norme presentate in questo documento hanno lo scopo di essere il più prescrittive possibile, al fine di definire in modo "algoritmico" le procedure di lavoro, e limitare fortemente lo spazio alle scelte di libero arbitrio che rischiano di portare a situazioni meno controllate rispetto a quelle previste durante la stesura del way of working.

Prima di leggere un qualunque documento prodotto da QB Software è necessario conoscere il significato dei termini riportati nel documento: *Glossario v2.0.0*; presente nella repo: Documentazione [Online - GitHub; ultima visita 18/11/2023].

1.2 Riferimenti

1.2.1 Normativi

- Standard ISO/IEC 12207:1997

  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf

  [Online - PDF; ultima visita 15/02/2024];

- Capitolato d'appalto C8:

  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C8.pdf

    [Online - PDF; ultima visita 30/11/2023];

  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C8p.pdf

    [Online - PDF; ultima visita 30/11/2023].

[1] Eccezione per la sezione: 1, A e B.

1.2.2 Informativi

- *Glossario dei termini v2.0.0*;
- *Piano di Qualifica v2.0.0*;
- Dispense dell'insegnamento d'Ingegneria del Software e approfondimenti:
  - Lezione T2 - Processi di ciclo di vita:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T2.pdf
    [Online - PDF; ultima visita 13/01/2024];
  - Lezione T4 - Gestione di progetto:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T4.pdf
    [Online - PDF; ultima visita 13/01/2024];
  - Lezione PD5 - Amministrazione di Progetto:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD5.pdf
    [Online - PDF; ultima visita 11/12/2023];
  - Lezione T7 - Qualità del software:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T7.pdf
    [Online - PDF; ultima visita 13/01/2024];
  - Lezione T10 - Verifica e Validazione - analisi statica:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T10.pdf
    [Online - PDF; ultima visita 02/02/2024];
  - Lezione T11 - Verifica e Validazione - analisi dinamica:
    https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T11.pdf
    [Online - PDF; ultima visita 02/02/2024].

2 Processi primari

I processi primari, indicati dallo standard ISO/IEC 12207:1997, sono processi che comprendono le attività direttamente legate allo sviluppo del software.

2.1 Processo di fornitura

Il processo di fornitura comprende le attività e i compiti del fornitore. Il processo ha come obiettivo determinare le procedure e le risorse necessarie per gestire e garantire il prodotto software all'acquirente. Il fornitore gestisce il processo di fornitura seguendo il processo di gestione organizzativa e il processo di formazione.
Come stabilito dallo standard ISO/IEC 12207:1997 il processo di fornitura identifica le seguenti attività:

1. avvio (sezione 2.1.1);
2. preparazione della risposta (sezione 2.1.2);
3. contrattazione;
4. pianificazione (sezione 2.1.3);
5. esecuzione e controllo (sezione 2.1.4);
6. revisione e valutazione (sezione 2.1.5);
7. consegna e completamento (sezione 2.1.6).

2.1.1 Avvio

Questa attività consiste nei seguenti compiti:

- il fornitore conduce una revisione dei vari capitolati d'appalto;
- il fornitore produce il documento *Valutazione dei capitolati*, che comprende per ogni capitolato valutato:
    1. una breve descrizione;
    2. il dominio applicativo;
    3. il dominio tecnologico;

4. gli aspetti positivi;
5. i fattori critici;
6. conclusioni.

2.1.2 Preparazione alla risposta

Il fornitore, dopo aver scelto il capitolato, deve produrre i seguenti documenti:

- la *Lettera di presentazione*, che comprende:
  1. il capitolato scelto;
  2. una lista dei documenti allegati;
  3. una lista dei membri del gruppo.
- il *Preventivo dei costi* e degli impegni, che comprende:
  1. gli impegni orari per ogni ruolo;
  2. una breve considerazione per ogni ruolo;
  3. preventivo dei costi, basato sulle tariffe orarie dei ruoli;
  4. scadenza di consegna;

2.1.3 Pianificazione

Questa attività consiste nella produzione dei seguenti documenti:

- il *Piano di Progetto*, che comprende:
  1. **analisi dei rischi**: analizza i rischi che possono influenzare la pianificazione del progetto, la qualità del software o portare al fallimento stesso del progetto;
  2. **modello di sviluppo**: analizza il modello scelto dal fornitore;
  3. **pianificazione temporale**: analizza la pianificazione temporale del progetto, e in particolare la suddivisione in fasi e la loro durata;
  4. **preventivi di periodo**: viene calcolata una stima del tempo di lavoro necessario e dei costi per ogni fase;

5. **consuntivi di periodo**: riporta le attività effettivamente completate e i costi reali, sia in termini economici che temporali;

- il *Piano di Qualifica*, che comprende:

1. **obiettivi di qualità**: riporta tutte le metriche utilizzate durante il progetto raggruppate per: metriche di processo e di prodotto;
2. **strategie di testing**: riporta tutte le strategie per garantire la qualità del prodotto, inoltre traccia lo stato d'implementazione di ogni singolo test;
3. **valutazione per il miglioramento**: analizza il processo di automiglioramento del gruppo QB Software;
4. **cruscotto**: traccia il valore di ogni metrica nel tempo, riportando eventuali considerazioni;

- l'*Analisi dei requisiti*, che comprende:

1. **introduzione**: viene descritto lo scopo del documento e lo scopo del progetto. Inoltre vengono forniti i riferimenti normativi e un glossario;
2. descrizione del prodotto;
3. analisi dei casi d'uso obbligatori;
4. analisi dei casi d'uso facoltativi.

2.1.4 Esecuzione e controllo

Questa attività consiste nei seguenti compiti:

- il fornitore deve implementare ed eseguire il *Piano di Progetto* (sezione 3.1.1);
- il fornitore deve sviluppare il prodotto software in conformità con il processo di sviluppo (sezione 2.2).

2.1.5 Revisione e valutazione

Questa attività consiste nell'eseguire la verifica e la validazione del prodotto software in conformità con il processo di verifica (sezione 3.4) e di validazione (sezione 3.5).

2.1.6 Consegna e completamento

Questa attività consiste nei seguenti compiti:

- il fornitore deve consegnare il prodotto software;
- il fornitore deve fornire la documentazione necessaria.

2.1.7 Strumenti

- **Google Caldendar**: utilizzato per i promemoria per i meeting con il proponente Zextras;
- **Google Slides**: utilizzato per creare le presentazioni PowerPoint dei diari di bordo;
- **Carbonio chat**: piattaforma per video chiamate sviluppata dal proponente utilizzata per i meeting;
- **Discord**: piattaforma in cui è stato creato un apposito canale di comunicazione condiviso con il proponente per questioni rapide;
- **GanttProject**: software utilizzato per generare i Gantt da inserire nel documento *Piano di Progetto*;
- **Google Sheets**: utilizzato per la creazione di grafici per il *Piano di Progetto* e il calcolo e i grafici delle metriche per il *Piano di Qualifica*.

2.2 Processo di sviluppo

Secondo lo standard ISO/IEC 12207:1997, lo scopo del processo di sviluppo è quello di definire le attività di analisi dei requisiti, progettazione, codifica, integrazione, testing, installazione e accettazione relativi al prodotto software. Lo sviluppatore che esegue o supporta tali attività, lo fa in conformità dei requisiti definiti da contratto.

2.2.1 Descrizione

Segue un elenco delle attività che caratterizzano il processo di sviluppo:

- analisi dei requisiti (sezione 2.2.2);
- progettazione architetturale (sezione 2.2.3);

- codifica (sezione 2.2.4).

2.2.2 Analisi dei requisiti

Scopo

Gli obiettivi dell'attività di analisi dei requisiti sono:

- concordare con il proponente lo scopo del prodotto da realizzare, rispecchiandone le specifiche;
- fornire ai progettisti dei requisiti chiari e ottimali;
- fornire una stima sulle tempistiche necessarie per completare il prodotto, in modo da favorire la pianificazione;
- favorire l'attività di verifica.

Descrizione

Il compito degli analisti è quello di effettuare l'analisi dei requisiti, redigendo un documento omonimo che deve contenere:

- **introduzione**: spiega lo scopo del documento;
- **descrizione**: enuncia le finalità del prodotto;
- **attori**: chi interagisce con il prodotto;
- **casi d'uso**$_G$: tutte le possibili interazioni che possono avvenire con il prodotto da parte degli attori;
- **requisiti**$_G$: le caratteristiche funzionali e non, da soddisfare.

Da questa attività, è previsto che venga redatta tutta la documentazione ufficiale che includa tutti i requisiti attesi dal cliente.

Casi d'uso

I casi d'uso sono un insieme di possibili sequenze d'interazioni compiute da uno specifico attore per raggiungere un particolare obiettivo all'interno del prodotto. I casi d'uso devono essere suddivisi in base alla tipologia di scenario che descrivono, in modo da

migliorare la chiarezza e la comprensione. All'interno della propria sezione in cui vengono descritti gli use case, ogni caso d'uso deve avere una *sotto sezione*$_G$ dedicata, in cui viene analizzato e suddiviso in sotto casi, nella maniera più specifica possibile.
Ogni caso d'uso deve essere costituito, in ordine, da:

- **Identificazione**: definita dal seguente formato

  `UC [numero_padre].[numero_figlio] - titolo`

  dove:

  - `UC`: è l'acronimo di Use Case;
  - `[numero_padre]`: numero identificativo del caso d'uso;
  - `[numero_figlio]`: numero identificativo progressivo dei sotto casi;
  - `titolo`: nome auto esplicativo del caso d'uso.

- **Diagramma**: strumento grafico utilizzato per rappresentare visivamente le interazioni tra attori e il sistema. È sufficiente un solo diagramma riassuntivo all'inizio dei sottocasi d'uso, per una dare una visione generale;
- **Attore principale**: enuncia l'entità che interagisce con il sistema;
- **Descrizione**: breve descrizione del caso d'uso;
- **Precondizioni**: le condizioni del sistema che devono essere soddisfatte prima che il caso d'uso possa essere eseguito;
- **Postcondizioni**: le condizioni del sistema che si verificano al termine dell'esecuzione del caso d'uso;
- **Scenario principale**: un elenco numerato che rappresenta il flusso degli eventi principali che si verificano durante l'esecuzione del caso d'uso;
- **Inclusioni**: (opzionale) indica che la funzione rappresentata da uno dei due use case include completamente la funzione rappresentata dall'altro;
- **Generalizzazioni**: (opzionale) usati per aggiungere caratteristiche e funzionalità rispetto ai padri, o modificarne il comportamento;

- **Estensioni**: (opzionale) impiegate per modellare scenari alternativi. Al verificarsi di una determinata condizione, il caso d'uso ad essa collegata viene interrotto;
- **Diagramma dei sotto-casi d'uso**: (opzionale) usato per visualizzare meglio i sotto-casi d'uso relativi del padre.

Di seguito, viene riportato un esempio di caso d'uso redatto secondo quanto normato (i diagrammi sono omessi per fornire maggiore chiarezza alla struttura):

UC 1 - Invio e-mail

- **Attore principale**: MUA;
- **Descrizione**: il MUA deve poter inviare una e-mail al destinatario indicato;
- **Precondizioni**: l'account che il MUA gestisce è registrato nel sistema, ha una connessione aperta con il sistema ed è autenticato;
- **Postcondizioni**: l'e-mail è stata consegnata con successo al destinatario ed è stata salvata nel sistema;
- **Scenario principale**:
  1. il MUA trasmette l'id dell'account (UC 1.1);
  2. il MUA trasmette l'id dell'e-mail (UC 1.2);
  3. il MUA trasmette il destinatario dell'e-mail (UC 1.3);
  4. il MUA trasmette il mittente dell'e-mail (UC 1.4);
  5. il sistema elabora l'inoltro.
- **Inclusioni**: nessuna;
- **Generalizzazioni**: nessuna;
- **Estensioni**: nessuna;
- **Diagramma dei sotto-casi d'uso**: (omesso).

Diagrammi dei casi d'uso

- **Attore**: un attore è un'entità esterna che interagisce con il sistema. Un attore può essere un utente, una persona, altri sistemi o componenti esterne. Gli attori sono rappresentati come uno stickman al di fuori del rettangolo che delimita il sistema e identificati da un'etichetta contenente il nome identificativo dell'attore;

Figura 1: Rappresentazione attore.

- **generalizzazione tra attori**: la generalizzazione tra attori rappresenta una relazione di ereditarietà e si ha quando si identifica un attore base con attributi e comportamenti comuni, e poi si definiscono attori specializzati che ereditano tali attributi e comportamenti dal primo attore. Questo concetto riflette la gerarchia degli attori e consente di modellare più facilmente i requisiti del sistema. Le generalizzazioni tra attori sono rappresentate da una freccia vuota che parte dall'attore figlio e arriva all'attore padre;

Figura 2: Rappresentazione generalizzazione tra attori.

- **caso d'uso**: un caso d'uso rappresenta una funzionalità che l'attore può eseguire con il sistema. I casi d'uso sono rappresentati con un ovale contenente una numerazione univoca (ad esempio UC x, dove $x \in \mathbb{N}$) e un titolo esplicativo della

funzione che rappresentano. Ogni caso d'uso deve essere collegato tramite una linea continua agli attori che hanno accesso a quella funzionalità;

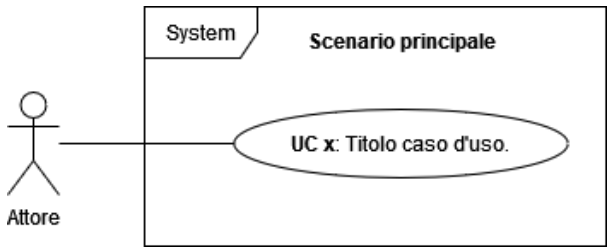

Figura 3: Rappresentazione caso d'uso.

- **sottocasi d'uso**: un sottocaso d'uso rappresenta un'analisi più dettagliata derivata da un caso d'uso più generale. I sottocasi d'uso sono rappresentati con un ovale contenente una numerazione univoca (esempio UC $x.y$, dove $x, y \in \mathbb{N}$) e un titolo esplicativo della funzionalità che rappresentano;

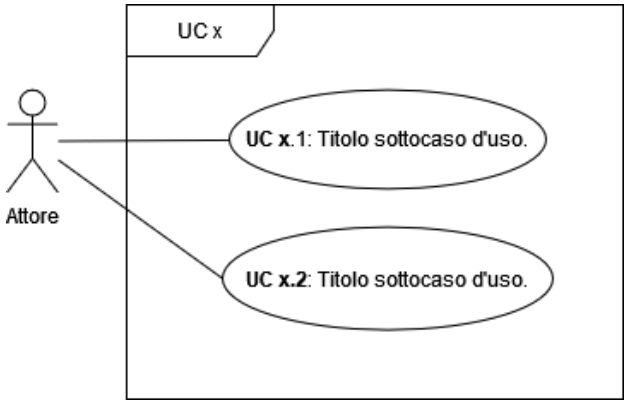

Figura 4: Rappresentazione sottocasi d'uso.

- **relazione di inclusione tra casi d'uso**: un'inclusione tra un caso d'uso A e un caso d'uso B si ha se tutte le istanze del caso d'uso A devono eseguire anche le istanze del caso d'uso B. Questa relazione permette di suddividere il comportamento complesso in parti più gestibili e di promuovere la riusabilità del codice. Le inclusioni sono rappresentate da una freccia tratteggiata con sopra scritta la direttiva `«include»`;

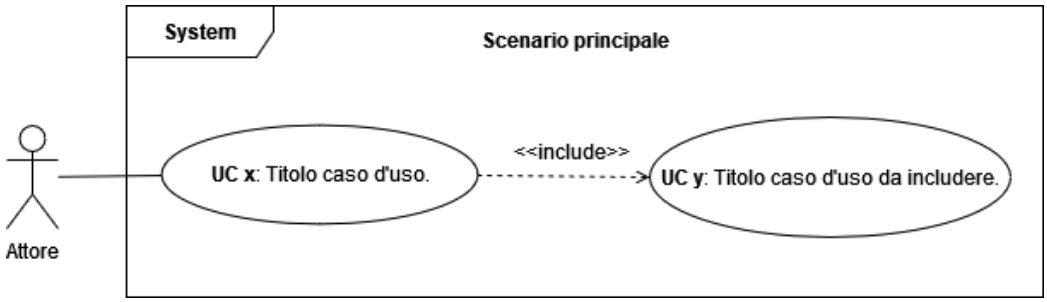

Figura 5: Rappresentazione relazione di inclusione.

- **relazione di estensione tra casi d'uso**: un'estensione tra un caso d'uso A e un caso d'uso B si ha quando il caso d'uso B fornisce un comportamento aggiuntivo che può essere attivato durante l'esecuzione del caso d'uso A, ma solo in determinate circostanze o condizioni. In altre parole, il caso d'uso B non è parte del flusso principale del caso d'uso A, ma viene invocato come una sorta di "estensione" solo se si verifica una determinata situazione o evento. Le estensioni sono rappresentate da una freccia tratteggiata con sopra scritta la direttiva `«extend»`;

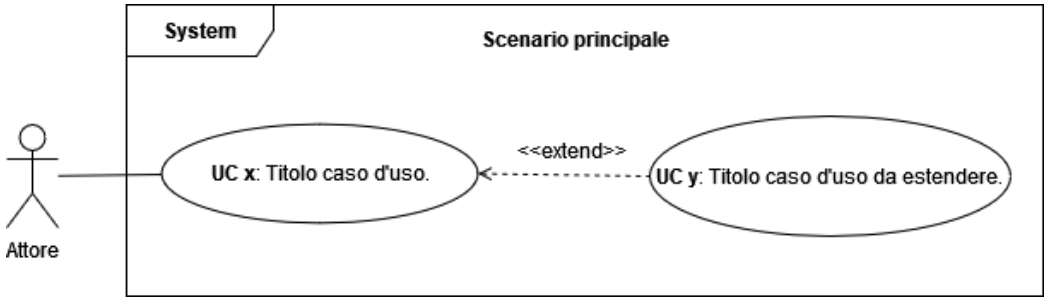

Figura 6: Rappresentazione relazione di estensione.

- **generalizzazione tra casi d'uso**: la generalizzazione tra casi d'uso indica che un caso d'uso più specifico eredita comportamenti da un caso d'uso più generale. Le generalizzazioni tra casi d'uso sono rappresentate da una freccia vuota che parte dal caso d'uso specifico e arriva al caso d'uso più generale;

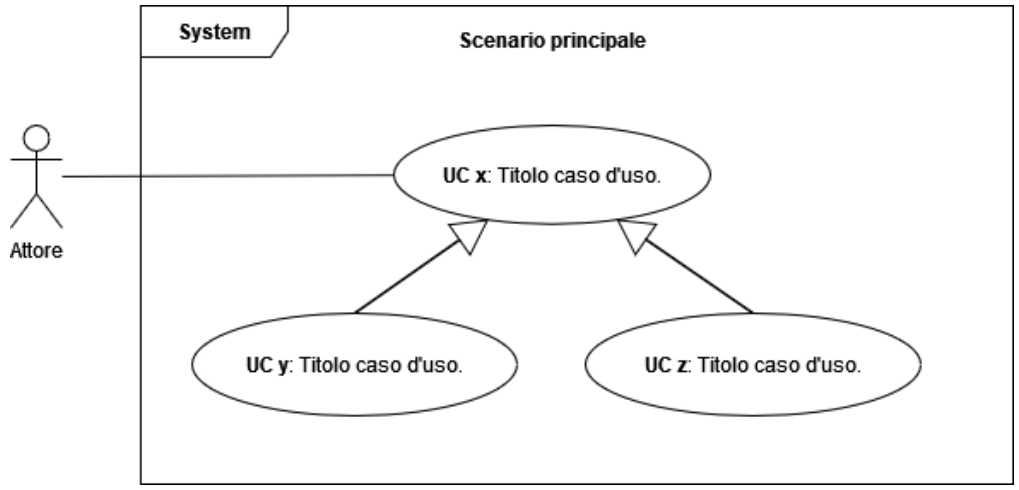

Figura 7: Rappresentazione generalizzazione tra casi d'uso.

Requisiti

Secondo la definizione dell'IEEE, i requisiti sono delle caratteristiche o delle proprietà che un certo prodotto software è tenuto ad avere da contratto. Nella redazione del documento di *Analisi dei Requisiti*, ogni requisito deve essere costituito da:

- **codice univoco**: che deve aderire al seguente formato:

  `R[Tipologia][Importanza] - [Codice]`

  dove:

  - `R`: sta per Requisito;
  - `[Tipologia]`: indica il tipo del requisito, che può assumere uno dei seguenti significati:
    - **F**: requisito **Funzionale**, con cui si descrivono i servizi e le funzioni che il sistema offre;
    - **V**: requisito di **Vincolo**, con cui si descrivono i vincoli ai servizi che il sistema offre;
    - **Q**: requisito di **Qualità**, con cui si descrivono i vincoli di qualità da realizzare secondo il *Piano di Qualifica*.
  - `[Importanza]`: indica il valore d'importanza che viene assegnato a ogni requisito; possono assumere i seguenti valori, decrescenti per valore d'importanza:

    * **O**: requisito **Obbligatorio** che deve essere necessariamente soddisfatto per garantire la presenza di funzionalità di base;
    * **D**: requisito **Desiderabile** che non ricopre una funzionalità fondamentale, ma che la sua implementazione da una maggiore completezza al prodotto;
    * **Z**: requisito **OpZionale** che determina ulteriore completezza all'interno del sistema. Rispetto ai precedenti, ha maggiore probabilità di comportare un dispendio di risorse.
  - `[Codice]`: identificatore numerico che associa ogni requisito al caso d'uso che lo implementa, seguendo la convenzione:

    `[codice_UC].[codice_sotto_UC]`

    dove:

    * `[codice_UC]`: indica l'identificatore numerico del caso d'uso base preso in esame;
    * `[codice_sotto_UC]`: (opzionale) indica l'identificatore progressivo relativo al sotto caso d'uso.

- **importanza**: che esprime in forma testuale uno dei tre valori d'importanza (obbligatorio, desiderabile, facoltativo). Nonostante questo dato sia ridondante, deve essere specificato per facilitare la lettura del documento;
- **descrizione**: fornisce una descrizione breve seppur completa, concentrandosi sulla chiarezza espositiva;
- **fonte**: indica il caso d'uso che lo implementa. Anche questo dato è ridondante, ma deve essere specificato per migliorare la tracciabilità.

Di seguito viene riportato un esempio, in forma tabellare, che rispetta quanto detto precedentemente:

Codice	Importanza	Descrizione	Fonte
RFO - 1	Obbligatorio	È necessario che il MUA abbia la capacità d'inviare un'e-mail	UC 1
RFO - 1.1	Obbligatorio	È necessario che il MUA abbia la capacità di trasmettere l'id dell'account durante l'attività di invio e-mail	UC 1.1

2.2.3 Progettazione architetturale

Definizione

Svolta da chi copre il ruolo di progettista. Consente di trasformare i requisiti in un'architettura che descrive una solida struttura e che identifica le componenti software. Questa operazione viene eseguita in maniera inversa rispetto a quella usata per l'analisi dei requisiti, infatti, se per l'analisi dei requisiti bisognava dividere un problema in parti per comprenderne il dominio applicativo, per la progettazione bisogna ricostruire il problema specificando le funzionalità di ogni parte. Deve assicurare che i requisiti siano prima soddisfatti e poi raffinati per facilitarne l'implementazione. Si compone di:

- **Technology Baseline**: la base tecnologica da cui iniziare la progettazione, contiene le specifiche ad alto livello del prodotto e delle sue componenti. Prevede inoltre una serie di diagrammi UML da cui è possibile costruire l'architettura e i test per la verifica. Essa prevede:
  - **tecnologie impiegate**: lista delle tecnologie scelte con le dovute motivazioni;
  - **Proof of Concept**: prototipo per dimostrare la fattibilità del prodotto software che si vuole realizzare.
- **Product Baseline:** il suo scopo è raffinare l'attività di progettazione, partendo dalla *Technology Baseline*. Grazie a questa è possibile definire i test per la verifica. Deve comprendere:
  - **definizione**: ogni classe è completa e definita. Ove nomi e funzionalità non sono ridondanti;

  - **diagrammi UML delle classi**: definiscono metodi, attributi e relazioni di una collezione di classi;
  - **design pattern**: giustificazione dei design pattern impiegati, specificandone significato e struttura;
  - **test di unità**: test effettuato su ogni singolo componente per verificarne la correttezza;
  - **tracciamento**: ogni requisito deve esser soddisfatto da una classe.

Output dell'attività

Ci si aspetta di:

- avere una buona comprensione di come si struttura il sistema;
- scegliere tecnologie efficienti ed efficaci determinate da un buono studio dei pro e i contro delle tecnologie prese in esame.

Documento

La progettazione produce il documento *Specifica Tecnica*, contenente:

- **tecnologie usate**: si descrivono le tecnologie utilizzate;
- **API**: si descrivono le API utilizzate;
- **architettura logica e di deployment**: si definiscono le componenti e le loro connessioni e come esse vengono distribuite nel sistema in esecuzione;
- **stato requisiti**: si riporta lo stato dei requisiti, indicando quali sono stati soddisfatti e quali no.

Qualità dell'architettura

Per assicurare che l'architettura del prodotto software sia di alta qualità adotteremo diverse best practice che coprono diverse fasi dello sviluppo del software. Tra questi, troviamo:

- **analisi dei requisiti**: assicurarsi di comprendere appieno i requisiti del sistema prima di iniziare a progettare l'architettura. Questo include la raccolta dei requisiti funzionali e non funzionali e la comprensione del contesto operativo del software;

- **design modulare e scalabile**: suddividere il sistema in moduli indipendenti e coesivi. Utilizzare principi di design come l'incapsulamento e l'astrazione per creare componenti che siano facilmente riutilizzabili e sostituibili. Inoltre, progettare l'architettura in modo che possa scalare in modo efficiente per gestire aumenti di carico o nuove funzionalità;
- **decoupling e coesione**: ridurre al minimo le dipendenze tra i moduli per rendere il sistema più flessibile e manutenibile. Utilizzare pattern come Dependency Injection e Event-Driven Architecture per ridurre l'accoppiamento. Allo stesso tempo, assicurarsi che i moduli siano coesi, cioè che ogni modulo abbia una singola responsabilità ben definita;
- **monitoraggio e analisi delle prestazioni**: incorporare strumenti di monitoraggio delle prestazioni nel sistema per identificare e risolvere eventuali problemi di prestazioni. Questo include il monitoraggio delle risorse di sistema, il rilevamento di eventuali falle di sicurezza e l'ottimizzazione delle query del database;
- **documentazione chiara e completa**: documentare l'architettura del software in modo chiaro e completo, inclusi diagrammi UML, descrizioni delle componenti e delle interazioni e spiegazioni delle decisioni di progettazione.

Diagrammi delle classi

I diagrammi delle classi si rendono necessari per definire le caratteristiche e le relazioni tra le componenti del sistema. Una classe viene rappresentata graficamente come dei rettangoli divisi in 3 parti:

1. **nome della classe**: nome identificativo della classe. Il nome è rappresentato in grassetto seguendo la convenzione PascalCase;
2. **attributi**: gli attributi seguiranno il seguente formato:

   ```
   visibilità nome:  tipo [molteplicità] = valori di default
   ```

   dove:

   - `visibilità`: precede obbligatoriamente ogni attributo e si riferisce alla capacità di altri componenti o parti del programma di accedervi. Può essere:
     - **-**: visibilità privata;

  - **+**: visibilità pubblica;
  - **#**: visibilità protetta;
  - **~**: visibilità di package.
- `nome`: nome univoco dell'attributo. Il nome è rappresentato in camelCase. Nel caso di attributo costante il nome verrà scritto interamente in maiuscolo;
- `molteplicità`: indica la lunghezza nel caso di una sequenza di elementi (come array o liste), si utilizza il seguente formato:

  `tipoAttrobuto [molteplicità]`

  Nel caso di lunghezza non conosciuta a priori, si utilizza:

  `tipoAttrobuto [*]`

  Nel caso di elemento singolo, la sua dichiarazione è opzionale;
- `valori di default`: ogni attributo può essere dichiarato con un valore di default;

3. **firme dei metodi**: descrivono il comportamento delle classi individuate. Le firme dei metodi seguiranno il seguente formato:

   `visibilità nome (parametri formali): tipo di ritorno`

   dove:

   - `visibilità`: corrisponde a quanto precedentemente descritto per gli attributi;
   - `nome`: nome univoco del metodo. Il nome è rappresentato in camelCase;
   - `parametri formali`: possono essere in numero da 0 a n e vengono separati da una virgola. Seguono le notazioni e le regole precedentemente descritti per gli attributi;
   - `tipo di ritorno`: determina che tipo di oggetto verrà ritornato dal metodo.

Relazioni tra classi

I diagrammi delle classi sono interconnessi da frecce che indicano le loro relazioni di dipendenza. Di seguito le possibili relazioni:

- **dipendenza**: una relazione di dipendenza tra classi si ha quando una classe, chiamata classe A, utilizza i servizi, i metodi o i membri forniti da un'altra classe, chiamata classe B. Questa relazione implica che la classe A dipende dalla classe B. In altre parole, se la classe B cambia. Questo tipo di relazione rappresenta il minimo grado di accoppiamento tra le due classi, in quanto la classe A è influenzata dalle modifiche alla classe B, ma non viceversa. Le relazioni di dipendenza sono rappresentate graficamente da una freccia tratteggiata che va dalla classe A alla classe B;

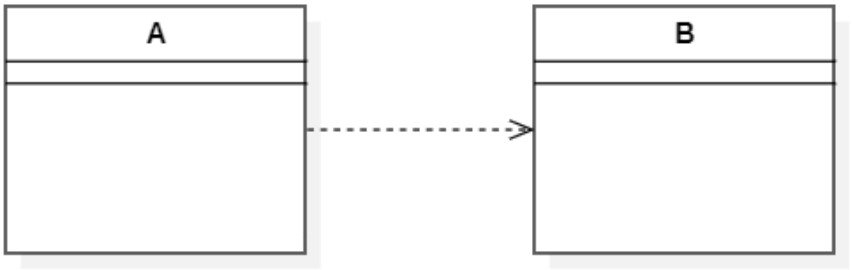

Figura 8: Rappresentazione relazione di dipendenza.

- **associazione**: una relazione di associazione si ha quanto una classe, chiamata classe A, contiene campi o istanze di un'altra classe, chiamata classe B. È possibile rappresentare le molteplicità di occorrenza tramite un valore posizionato agli estremi della freccia, il quale può essere:
  - **0..1**: A possiede 0 o 1 istanza di B;
  - **0..***: A possiede 0 o più istanze di B;
  - **1**: A possiede un'unica istanza di B;
  - *****: A possiede più istanze di B;
  - **n**: A possiede esattamente n istanze di B.

  Le relazioni di associazione sono rappresentate graficamente da una freccia continua che va dalla classe A alla classe B;

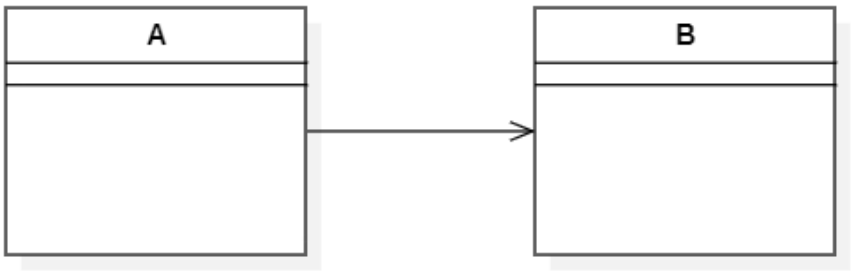

Figura 9: Rappresentazione relazione di associazione.

- **aggregazione**:una relazione di aggregazione si ha quando una classe, chiamata classe A, è composta da una o più altre classi, nell'esempio B, ma queste classi possono esistere indipendentemente dalla classe aggregante. Le relazioni di aggregazione sono rappresentate graficamente da una freccia a diamante vuota;

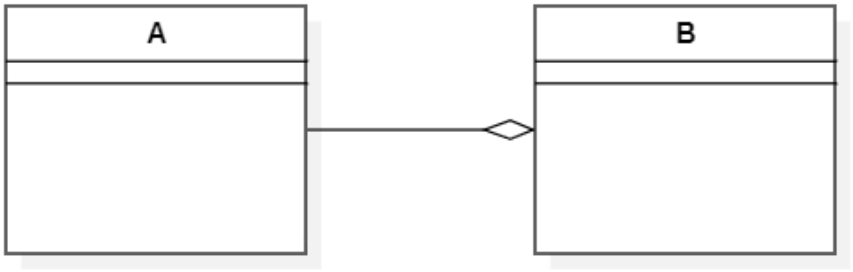

Figura 10: Rappresentazione relazione di aggregazione.

- **composizione**: una relazione di composizione si ha quando abbiamo una forma più forte di aggregazione, dove le classi aggregate (classe B) sono strettamente legate alla classe composita (classe A) e non possono esistere senza di essa. Le relazioni di composizione sono rappresentate graficamente da una freccia a diamante piena;

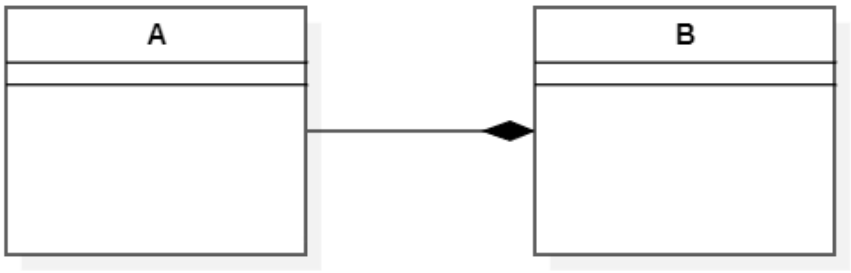

Figura 11: Rappresentazione relazione di composizione.

- **generalizzazione**: una relazione di generalizzazione si ha quando ogni oggetto di una classe B fa anche parte di una classe A. Tale relazione rappresenta il massimo grado di accoppiamento tra due classi. Le relazioni di generalizzazione sono rappresentate graficamente da una freccia continua vuota che va dalla classe B alla classe A.

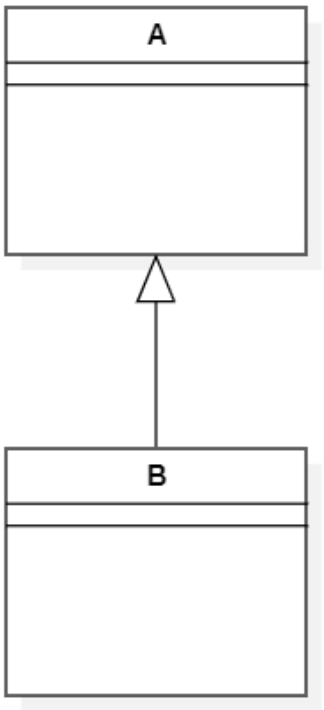

Figura 12: Rappresentazione relazione di generalizzazione.

- **interface relization**: una relazione di interface generalization si ha quando un'in-

terfaccia A può essere implementata utilizzando una classe B. Le relazioni di interface realization sono rappresentate graficamente da una linea continua che va dalla classe B alla classe A, la quale sarà rappresentata come un cerchio.

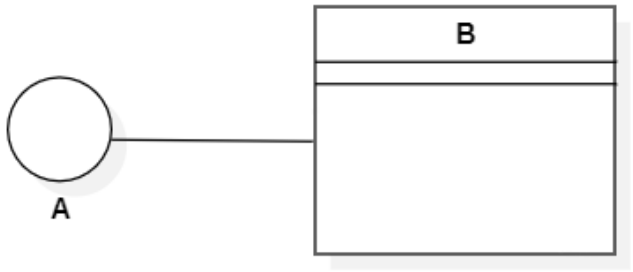

Figura 13: Rappresentazione relazione di interface realization.

Design pattern

Un design pattern è una soluzione generale e riconosciuta a un problema ricorrente nel campo della progettazione del software. Si tratta di un modello che fornisce una guida su come risolvere un determinato problema di progettazione, offrendo una struttura organizzata e ben testata che può essere riutilizzata in diverse situazioni. I design pattern sono sviluppati sulla base dell'esperienza pratica e sono progettati per migliorare l'efficienza, la qualità e la manutenibilità del software. Essi rappresentano una sorta di "best practice" che può essere applicata per affrontare specifiche sfide di progettazione.

Ogni design pattern deve includere una descrizione dell'importanza e dell'utilità del design pattern all'interno della struttura progettata, in modo da comprendere appieno il suo ruolo e la sua utilità all'interno dell'architettura complessiva.

Test

L'attività di testing svolge un ruolo cruciale nell'assicurare la qualità del prodotto finale. Durante questa fase, vengono delineati i requisiti di testing, definiti i casi di test e i criteri di accettazione, che fungono da strumenti per valutare il software. L'obiettivo primario è individuare e correggere eventuali difetti nel software prima del suo rilascio finale. Inoltre, il processo di testing è essenziale per garantire che il software soddisfi le specifiche e le aspettative del cliente.

Questa attività viene svolta dai progettisti. La sezione 3.4 fornisce una descrizione dettagliata delle diverse tipologie di test.

2.2.4 Codifica

Scopo

Svolta da chi copre il ruolo di programmatore. La stesura del codice viene realizzata secondo le indicazioni ricavate dal prodotto del lavoro dei progettisti. Il suo scopo è quello di realizzare il codice del prodotto software garantendo i seguenti punti:

- consistenza interna ed esterna al prodotto software;
- test di unità;
- uso appropriato dei metodi di codifica e degli standard a cui ci si appoggia;
- fattibilità dell'integrazione software coi dovuti test;
- il prodotto ultimo deve poter essere funzionale e manuntenibile.

Descrizione

Il prodotto dell'attività di codifica deve adempiere alle richieste del proponente e soddisfare i requisiti concordati. Perseguire le best practices in fase di codifica permette di:

- rendere il codice comprensibile, robusto agli errori, efficace ed efficiente;
- migliorare la qualità del prodotto.

Dai precedenti punti ne deriva una significativa facilitazione nelle attività di verifica, manutenzione ed estensione.

Stile di codifica

Ogni programmatore è tenuto a seguire le presenti norme in fase di codifica:

- **indentazione**: mantenere un'indentazione di quattro spazi come carattere per ogni sorgente;
- **costanti**: sono immediatamente riconoscibili poiché il loro nome si compone di solo di parole scritte in lettere maiuscole e sparate da trattini bassi;

- **classi**: il nome di una classe deve essere sempre scritto in *camel case*$_G$;
- **metodi**: il nome di un metodo deve essere sempre camel case per facilitarne la lettura. É inoltre opportuno mantenere il corpo dei metodi pulito, ridotto e comprensibile;
- **lingua**: la lingua adottata per l'assegnazione dei nomi a variabili e funzioni è quella inglese. I commenti al codice e i valori delle stringhe possono tuttavia essere espressi in italiano;
- **univocità dei nomi**: a eccezione di possibili overriding e/o overloading dei metodi non è possibile usare nomi identici per le funzioni. Inoltre variabili presenti nello stesso contesto devono avere nomi differenti fra loro indiscriminatamente dal tipo a esse assegnato;
- **ricorsione**: l'utilizzo dell'iterazione è preferibile in quanto rende la verifica del codice meno complessa;
- **complessità ciclomatica**: ogni metodo deve avere il minor numero possibile di cicli annidati. Massimo valore tollerabile: 5 (sezione B);
- **code coverage:** le aspettative di code coverage si reputano accettabili qualora esse siano superiori all'80% (sezione B).

3 Processi di supporto

I processi di supporto indicati dallo standard ISO/IEC 12207:1997 sono otto; un processo di supporto ha l'obiettivo di supportare altri processi, ad esempio quelli primari, con il fine di contribuire al successo e alla qualità di un progetto software.

3.1 Processo di documentazione

Il processo di documentazione ha lo scopo di registrare tutte le informazioni prodotte dal ciclo di vita di un processo o di un'attività. Di seguito riportiamo le quattro attività fondamentali della documentazione: implementazione del processo, design e sviluppo, produzione e manutenzione.

3.1.1 Implementazione del processo

In questa attività l'obiettivo è quello di pianificare lo sviluppo dei documenti, riportando le seguenti informazioni per ogni documento da produrre durante il progetto: il titolo del documento, il nome del documento in produzione, lo scopo, il target (a chi è rivolto il documento), le procedure che devono essere attuate, le responsabilità per l'attuazione delle procedure e le date/tempi previsti per il rilascio del documento.
Utilizzeremo il formato `YYYY-MM-DD` quando vogliamo indicare una data scritta nel seguente modo: anno-mese-giorno. Utilizzeremo il formato `X.Y.Z` per le versioni come riportato nella sezione 3.1.2. I documenti pianificati sono:

Lettera di presentazione

1. **Nome file**: `Lettera_di_presentazione_X.Y.Z.pdf`;
2. **scopo**: presentarsi alla candidatura di una milestone "esterna" (candidatura, RTB, PB o CA), riporta i documenti e le decisioni più rilevanti;
3. **target**: committente, documento esterno;

4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Responsabile	3.1.2
Manutenzione	Responsabile	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento deve essere redatto e approvato entro un periodo di tempo che precede di almeno tre giorni la scadenza fissata per la consegna della milestone.

Preventivo dei costi e degli impegni

1. **Nome file**: `Preventivo_dei_costi_e_degli_impegni_X.Y.Z.pdf`;
2. **scopo**: presentarsi alla prima milestone (candidatura), riporta i costi e la suddivisione delle ore per ruolo preventivate;
3. **target**: committente, documento esterno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Responsabile	3.1.2
Manutenzione	Responsabile	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: consegna entro il 31/10/2023 alle ore 17:00.

Valutazione capitolati

1. **Nome file**: `Valutazione_dei_capitolati_X.Y.Z.pdf`;
2. **scopo**: presentarsi alla candidatura, valutare i 3 capitolati che risultano essere i nostri preferiti tra quelli proposti, riportando quelle che sono state le opportunità e le criticità individuate dal team durante la scelta dei capitolati;
3. **target**: committente, documento esterno;
4. **procedura e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Team QB Software	3.1.2
Manutenzione	Team QB Software	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: consegna entro il 31/10/2023 alle ore 17:00.

Norme di progetto

1. **Nome file**: `Norme_di_progetto_X.Y.Z.pdf`;
2. **scopo**: normare il way of working in modo prescrittivo, regolando i vari processi proposti dello standard ISO del 12207 del 1997, basandosi sulle decisioni prese durante le riunioni;
3. **target**: team di QB Software, documento interno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Team QB Software	3.1.2
Manutenzione	Team QB Software	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento non prevede una versione finale in quanto è continuamente soggetto a modifiche e revisioni.

Glossario

1. **Nome file**: `Glossario_X.Y.Z.pdf`;
2. **scopo**: riportare tutti i termini di dominio utilizzate durante il progetto;
3. **target**: tutte le persone che devono interagire con questo progetto, documento esterno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Team QB Software	3.1.2
Manutenzione	Team QB Software	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento non prevede una versione finale in quanto è continuamente soggetto a modifiche e revisioni.

Verbali interni

1. **Nome file**: `Verbale_interno_YYYY-MM-DD_X.Y.Z.pdf`;
2. **scopo**: riportare le decisioni prese durante le riunioni interne ufficiali. Il documento ha come primo obiettivo quello di riportare le decisioni di pianificazione prese, fornendo contestualmente le ragioni di tali scelte, i ticket inseriti nel ITS dovuti ai compiti assegnati e gli argomenti da trattare per la prossima riunione;
3. **target**: team di QB Software, documento interno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Team QB Software	3.1.2
Manutenzione	Team QB Software	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il verbale deve essere redatto e verificato entro cinque giorni dall'avvenuta riunione.

Verbali esterni

1. **Nome file**: `Verbale_esterno_YYYY-MM-DD_X.Y.pdf`;
2. **scopo**: riportare le decisioni prese durante le riunioni esterne ufficiali. Il documento ha come primo obiettivo quello di riportare gli argomenti di discussione durante la riunione e i ticket inseriti nel ITS dovuti alle decisioni;
3. **target**: QB Software e partecipanti esterni, documento esterno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Team QB Software	3.1.2
Manutenzione	Team QB Software	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Approvazione (esterni)	Esterni	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il verbale deve essere redatto e verificato entro cinque giorni dall'avvenuta riunione, poi l'approvazione passa a tutti i proponenti esterni.

Piano di Qualifica

1. **Nome file**: `Piano_di_qualifica_X.Y.Z.pdf`;
2. **scopo**: normare le procedure di verifica con l'obiettivo di mantenere una qualità del prodotto alta;
3. **target**: QB Software, documento interno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Verificatore	3.1.2
Manutenzione	Verificatore	3.1.4
Verifica	Verificatore (che non ha redatto la modifica)	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: la prima versione deve essere rilasciata entro l'ultima settimana di novembre 2023, ulteriori aggiornamenti sono previsti.

Analisi dei requisiti

1. **Nome file**: `Analisi_dei_requisiti_X.Y.Z.pdf`;
2. **scopo**: contiene i requisiti individuati e gli use case per il prodotto da sviluppare;
3. **target**: QB Software, documento esterno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Analisti	3.1.2
Manutenzione	Analisti	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: una prima versione del documento deve essere prodotta entro il primo incontro con il proponente, dove si deve parlare dei requisiti, la prima versione deve essere messa in produzione entro RTB.

Piano di Progetto

1. **Nome file**: `Piano_di_progetto_X.Y.Z.pdf`;
2. **scopo**: contiene la pianificazione del lavoro, come vengono allocate le risorse, gli obiettivi e come essi vengono raggiunti;
3. **target**: QB Software, documento esterno;
4. **procedure e responsabilità**:

Procedura	Responsabilità	Attività di riferimento (NdP)
Sviluppo	Responsabile	3.1.2
Manutenzione	Responsabile	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento viene aggiornato a ogni Sprint.

Specifica Tecnica

1. **Nome file**: `Specifica_Tecnica_X.Y.Z.pdf`;
2. **scopo**: contiene l'architettura logica del prodotto, l'architettura di deployment e i design pattern;
3. **target**: QB Software, documento esterno;
4. **procedure e responsabilità**:

Procedura	Responsabilità	Attività di riferimento (NdP)
Sviluppo	Progettista	3.1.2
Manutenzione	Progettista	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento viene aggiornato ad ogni cambiamento nella progettazione.

Manuale Utente

1. **Nome file**: `Piano_di_progetto_X.Y.Z.pdf`;
2. **scopo**: contiene le istruzioni per l'uso e il funzionamento del prodotto;
3. **target**: QB Software, documento esterno;
4. **procedure e responsabilità**:

Procedura	**Responsabilità**	**Attività di riferimento (NdP)**
Sviluppo	Programmatore	3.1.2
Manutenzione	Programmatore	3.1.4
Verifica	Verificatore	3.1.2
Approvazione	Responsabile	3.1.3
Configuration managment	Amministratore	3.1.2
Produzione e distribuzione	Responsabile	3.1.3

5. **schedule**: il documento viene aggiornato a ogni funzionalità aggiunta.

3.1.2 Design e development

In questa attività vengono riportati tutti gli strumenti necessari allo sviluppo della documentazione, vengono definite le regole d'impaginazione dei documenti da produrre. Lo scopo è quello di partire dai sorgenti .tex per ottenere dei documenti in formato PDF che seguano le scelte tipografiche scelte dal team QB Software.

Strumenti per lo sviluppo

- I documenti devono essere scritti in LaTeX, usando la distribuzione TexLive [Online; ultima visita 18/11/2023];
- non viene imposto nessun vincolo per l'editor/IDE da utilizzare per scrivere i documenti, comunque, si consiglia di utilizzare Visual Studio Code [Online; ultima visita 18/11/2023] con l'estensione LaTeX Workshop [Online; ultima visita 18/11/2023];
- il VCS da utilizzare per tracciare la storia dello sviluppo dei documenti è Git [Online; ultima visita 18/11/2023];
- ogni documento deve importare il sorgente LaTeX `base.tex`, il quale contiene tutte le utilità e le regole tipografiche normate in questo documento per lo sviluppo della documentazione;
- vengono messi a disposizione i seguenti template, presenti nel repository al path `templates/`:
  - la cartella `empty/`, struttura di un documento di base generico, da questo template derivano tutti gli altri template;
  - la cartella `verbale_interno/`, struttura di un documento per i verbali interni;
  - la cartella `verbale_esterno/`, struttura di un documento per i verbali esterni.
- GitHub [Online; ultima visita 18/11/2023] come servizio per:
  - creare una repository privata per il tracciamento della storia dei sorgenti dei documenti, nel repository `docs` ramo `develop` dove vengono caricati i documenti verificati, ma non ancora approvati;
  - pubblicare una repo pubblica per consegnare i documenti approvati al committente.

Impaginazione di base

Ogni documento di QB Software deve essere sviluppato a partire da un template di base, presente nella cartella `src/templates/empty` nel ramo di `develop`. Il template di base deve rispettare la seguente impaginazione:

I) deve essere in formato A4, dimensione font 11pt, margine di 2.5 cm;

II) la prima pagina deve riportare nel seguente ordine:

1. la scritta "QB Software";
2. il logo di QB Software;
3. il logo dell'università di Padova;
4. la scritta "Università degli studi di Padova";
5. la scritta "corso di ingegneria del software";
6. la scritta "anno accademico 2023/2024";
7. il titolo del documento, e quando richiesto anche la data;
8. il contatto e-mail di QB Software come link;

III) la seconda pagina è dedicata al registro delle modifiche descritto al paragrafo 3.1.2;

IV) una pagina dedicata all'indice dei contenuti generato da LaTeX.

Ogni documento deve riportare su ogni pagina, a eccezione della prima pagina, un piè di pagina e una testatina separate dal contenuto con una linea. In ogni testatina deve essere riportato nel margine destro il logo del gruppo e nel margine sinistro la scritta "QB Software". Ogni piè di pagina deve riportare nel margine sinistro il titolo del documento e nel margine destro la pagina attuale nella seguente forma: `Pagina x di y`, dove `x` è la pagina attuale, e `y` il totale delle pagine senza contare la prima.

Regole tipografiche e di coding

Di seguito ridefiniamo, o aggiungiamo, ulteriori regole tipografiche oltre a quelle normalmente usate dal LaTeX, con lo scopo di rendere il documento più accessibile, ed evitare incongruenze di stile tra i diversi documenti da produrre:

- ogni tabella e figura libera presenti nel documento, a eccezione del *registro delle modifiche* e dei loghi, devono essere accompagnati da una didascalia che ne descrive il contenuto. A questo scopo è necessario usare l'ambiente LaTeX `figure` o `table` e l'istruzione `\caption` per la didascalia;
- ogni tabella e figura deve inoltre essere dotata di una label, creata con il commando `\label`. Le label devono iniziare come *fig:* per le figure, e *table:* per le tabelle;

- le tabelle vanno inserite in un ambiente `table` e devono essere posizionate sempre all'inizio della pagina, come da impostazione predefinita per l'ambiente citato, possono fare eccezione dei casi particolari dove la presenza della tabella in un certo punto del testo permette una migliore comprensione del discorso;
- quando ci si riferisce a una figura, una tabella, oppure a una sezione, citarla con il comando `\ref` specificando la tipologia (tabella, figura, sezione) dell'elemento citato seguito dal numero della sezione; Esempio:

  [...] le regole tipografiche sono riportate nella sezione 3.1.2 [...]

- ogni link deve essere inserito sotto forma di testo sottolineato di colore blu, inoltre non si deve scrivere direttamente l'URL, ma una frase chiara che specifichi dove quel link stia puntando;
- soltanto i link posso essere sottolineati, nessun'altra parte del testo può essere sottolineata;
- ogni sezione creata con il commando `\section` deve iniziare sempre in una nuova pagina, per fare ciò ogni section di testo va scritta in un file .tex a parte, sotto la cartella `sections/` e importando nel documento principale attraverso il comando `\include`;
- i riferimenti a immagini, sezioni e tabelle, interni al documento vengono colorati di azzurro, questo colore non deve essere utilizzato per nessun'altra parte del testo che non sia un link.

Registro delle modifiche

Il registro delle modifiche, per tenere una traccia completa e sensata della storia del documento deve riportare i seguenti dati:

1. versione del documento;
2. data della modifica;
3. membro del gruppo che ha introdotto la modifica;
4. ruolo assunto dall'autore al momento della stesura;
5. chi si è occupato della verifica (implicitamente il ruolo sarà sempre verificatore);

6. data del superamento della verifica;
7. descrizione, breve, ma significativa delle modifiche apportante, con riferimento alla sezione modificata o introdotta.

Il registro delle modifiche deve essere implementato attraverso l'ambiente LaTeX `changelog` definito all'interno del pacchetto `base.tex`. Tale ambiente deve provvedere a creare:

1. il titolo "Registro delle modifiche", il quale non verrà riportato nell'indice del documento;
2. una tabella formata dalle seguenti quattro colonne, nel seguente ordine:
   (a) **V.**: vengono riportate le versioni del documento al momento dell'approvazione della modifica;
   (b) **data**: vengono riportate le date di stesura della modifica e di approvazione da parte del verificatore;
   (c) **membro**: vengono riportati gli autori della modifica, e i verificatori che hanno approvato la modifica;
   (d) **ruolo**: vengono riportati i ruoli degli autori al momento della modifica, mentre per chi ha fatto la verifica viene riportato il ruolo di verificatore;
   (e) **descrizione**: vengono riportate le modifiche, o aggiunte, fatte al documento facendo riferimento alle sezioni che hanno subito la modifica, o aggiunta.

Inoltre il sorgente `base.tex` fornisce il comando:

```
\newlog{Ver}{DataModifica}{Membro}{RuoloMembro}
    {DataVerifica}{Verificatore}{Descizione}
```

che permette d'inserire una nuova modifica all'interno del registro delle modifiche. Il comando deve essere usato all'interno dell'ambiente `changelog` dentro il pacchetto precedentemente citato.

Versioni documenti

La versione dei documenti proposta è composta da tre cifre:

$$x.y.z$$

- x indica l'approvazione e il rilascio del documento per una milestone esterna (RTB, PB o CA);
- y rappresenta una modifica sostanziale, come l'aggiunta di una nuova sezione;
- z rappresenta una modifica minore, come l'aggiornamento di una paragrafo;

Sviluppo dei documenti

Di seguito illustriamo le fasi per lo sviluppo di ogni documento. Ogni creazione/modifica di un documento deve essere collegata a una issue in modo da rendere tracciabile il lavoro svolto, vedere la sezione 3.1.2: automazione build per i documenti. QB Software ha individuato due momenti differenti del ciclo di vita di un documento: la creazione del documento, distinta da continue aggiunte di argomenti nuovi e sostanziali aggiunte in termini di contenuto, e la manutenzione del documento, distinta dall'assenza di modifiche sostanziali al contenuto e da modifiche che mirano a correggere o rendere il documento più fruibile.
Riportiamo la procedura per lo sviluppo di un documento:

1. il membro che deve scrivere la modifica crea un nuovo branch a partire da `develop` usando la sezione *Development* nel form della issue su GitHub;
2. i redattori sviluppano la parte richiesta seguendo le regole imposte dalle norme di progetto, ogni volta che terminano una parte del lavoro pubblicano sul proprio ramo le modifiche in modo da renderle disponibili a tutto il team di QB Software. In questa fase il documento viene considerato ancora una bozza;
3. quando un relatore ha finito la task e il documento è pronto, aggiorna il registro delle modifiche e richiede una pull request del proprio branch con il `develop`, e assegna un verificatore che dovrà verificare il lavoro;
4. il verificatore assegnato dovrà verificare i contenuti secondo quanto riportato nella sezione 3.1.2;
5. il verificatore se approva il contenuto deve fare il merge della pull request e elimina il branch creato dalla issue, in caso di rifiuto il redattore deve assolvere alle mancanze e iniziare nuovamente la procedura di verifica.

Verifica del documento

La verifica del documento per la fase di creazione di un documento viene fatta attraverso il metodo del *walkthrough*, cioè il verificatore legge l'intero documento e controlla:

- il pieno rispetto delle scelte tipografiche dettate dalle norme di progetto;
- la soddisfazione delle modifiche richieste dalla issue.

La verifica del documento per la fase di manutenzione avviene attraverso delle *checklist* presenti nelle Piano di Qualifica. Terminata la procedura di verifica il verificatore deve pubblicare un commento sulla pull request riportando il feedback della verifica.

Automazione build per i documenti

La configurazioni per l'automazione della build di ogni documento sono contenute dentro il file `config.csv`. Il file di configurazione appena citato deve essere gestito dall'amministratore. Oltre al file di configurazione il sistema di build usa le label assegnate alla pull request (che devono essere le stesse label assegnate alla issue) per impostare ulteriori parametri di compilazione. È responsabilità dell'autore della modifica e del verificatore nel assegnare le label corrette alla issue e alla pull request. Per la compilazione dei documenti le label da inserire sono:

1. **documentation**: indica che la pull request sta per introdurre un documento da compilare;
2. **enhancement** o **maintenance**: nel primo caso se la modifica è un aggiunta sostanziale, nel secondo caso se la modifica è di manutenzione. Permette di determinare come avviene l'aggiornamento della versione;
3. **<Sigla Documento>**: serve a identificare quale documento costruire, es: NdP (Norme di Progetto), PdP (Piano di Progetto), e così via.

3.1.3 Produzione

Mettere in produzione i documenti

Il responsabile può provvedere al rilascio di un documento da `develop` al `main` attraverso una pull request. L'approvazione e il conseguente rilascio di un documento deve essere fatto dal responsabile solo quando il documento è pronto per la pubblicazione per la

milestone esterna. Il documento è disponibile nel branch `main` e nel sito web di QB Software [Online; ultima visita 18/11/2023].

Mettere in produzione i documenti validati da esterni

I documenti che devono essere validati dagli esterni richiedono prima di seguire la procedura indicata nella sezione 3.1.3 il seguente requisito:

- una firma, o una qualunque marcatura che possa dimostrare l'approvazione da parte degli esterni.

Pubblicazione

I documenti verificati ma non approvati verranno pubblicati nel `develop` della repository privata *docs* di QB Software, una volta approvati dal responsabile tramite pull request vengono rilasciati nel branch `main`. I documenti approvati verranno poi pubblicati nel `main` della repository ufficiale *Documentazione* di QB Software e nel sito di QB Software [Online; ultima visita 18/11/2023].

3.1.4 Manutenzione

Ogni documento che necessità di manutenzione può avere due tipi di modifiche:

- **di correzione**: vengono corrette alcune parti del documento senza modificarne la struttura;
- **di refactoring**: viene modificata la struttura del documento, senza l'aggiunta di contenuti.

3.2 Processo di gestione della configurazione

Il processo di gestione della configurazione si propone di assicurare che il software venga sviluppato, modificato e rilasciato in modo ordinato e controllato lungo tutto il suo ciclo di vita.
Questo processo si propone di raggiungere diversi obiettivi:

- identificare e definire tutti gli elementi software nel sistema;
- gestire le modifiche e i rilasci del software in modo organizzato e approvato;

- tenere traccia dello stato attuale degli elementi e delle richieste di modifica;
- garantire che tutti gli elementi del software siano completi, coerenti e corretti;
- assicurare che gli elementi software siano conservati, gestiti e consegnati in modo sicuro e controllato.

Il processo si compone delle seguenti attività:

- implementazione di processo;
- identificazione della configurazione;
- controllo della configurazione;
- stato della configurazione;
- valutazione della configurazione;
- controllo del rilascio.

3.2.1 Implementazione del processo

In questa attività vengono definite le strategie, le procedure e le linee guida per la gestione efficace della configurazione di un progetto.

3.2.2 Identificazione della configurazione

Per garantire un'identificazione ottimale di ciascun elemento software, vengono di seguito descritte le tecnologie e le metodologie da noi adottate.

Tecnologie adottate

- **Git**: utilizzato per il versionamento del codice sorgente, sia software che documentazione;
- **GitHub**: utilizzato come Issue Tracking System.

Lista repository

Vengono utilizzate 3 repository:

- Documentazione [Online; ultima visita 11/12/2023]: repo pubblica contenente i file da consegnare al committente;
- docs [Online; ultima visita 11/12/2023]: repo privata contenente la documentazione interna ed esterna;
- PoC [Online; ultima visita 11/12/2023]: repo pubblica contenente il Proof of Concept.

Gerarchia dei file

La repository nominata docs è composta dalle seguenti cartelle:

- **Candidatura**: contenente i documenti di candidatura al progetto didattico redatti dal gruppo;
- **RTB**: contenente i documenti da consegnare alla revisione RTB, quando saranno disponibili. Tale cartella sarà così suddivisa:
  - Documenti esterni: contenente i documenti ad uso esterno, in particolare:
    * Analisi dei Requisiti v2.0.0;
    * Piano di Progetto v2.0.0;
    * Piano di Qualifica v2.0.0;
    * una cartella Verbali, la quale contiene i verbali esterni.
  - Documenti interni: contenente i documenti ad uso interno, in particolare:
    * Glossario v2.0.0;
    * Norme di Progetto v2.0.0;
    * una cartella Verbali, la quale contiene i verbali interni.
- **PB**: contenente i documenti da consegnare alla revisione PB, quando saranno disponibili. Tale cartella sarà così suddivisa:
  - Documenti esterni: contenente i documenti ad uso esterno, in particolare:
    * Analisi dei Requisiti v2.0.0;
    * Glossario v2.0.0;

        * Piano di Progetto v2.0.0;
        * Piano di Qualifica v2.0.0;
        * una cartella Verbali, la quale contiene i verbali esterni.
    - Documenti interni: contenente i documenti ad uso interno, in particolare:
        * Norme di Progetto v2.0.0;
        * una cartella Verbali, la quale contiene i verbali interni.

- **site**: contenente il codice per la gestione del sito.

Branch

Il flusso di lavoro adottato prevede l'utilizzo di più branch. Prima di essere incluse nel branch `main`, le modifiche devono essere verificate e approvate. Tutto ciò che ancora non è stato approvato ma è stato verificato si trova nel branch `develop`. Ciascuna issue genererà la creazione di un branch a partire da `develop`. Successivamente, quando un compito sarà pronto per la verifica si aprirà una `pull request` per includere le modifiche nel branch `develop`.
Questa procedura è descritta dettagliatamente nella sezione 4.1.3.

A ridosso delle due revisioni previste (RTB e PB), quando i file saranno pronti per l'approvazione, si aprirà una `pull request` dal branch `develop` verso il branch `main` per poter integrare le modifiche.

Versionamento nel nome dei file

Ogni documento PDF aggiunto alla repository, in seguito a una pull request riuscita, possiede nel nome già la versione corrente. Questo grazie a uno script Python ad hoc (descritto alla sezione 3.1.2) che permette di eseguire questa operazione automaticamente tramite le GitHub Actions.

3.2.3 Controllo della configurazione

Per garantire un adeguato controllo delle modifiche, sarà seguito il procedimento di seguito descritto. Si noti che tale procedimento è applicato per modifiche sostanziali ai documenti e al codice. Per le modifiche minori tutti i membri del gruppo hanno il permesso di apportare modifiche ai file.

Richieste di modifica

Durante le riunioni interne, ogni membro del team ha la possibilità di proporre modifiche al software. Nel caso in cui tale richiesta risulti essere adeguata, verrà riportata nel verbale della riunione in cui viene presentata.

Analisi e valutazione modifiche

Durante la riunione in cui una richiesta di modifica viene proposta, se possibile, verrà valutata immediatamente. In caso contrario, se la richiesta richiede un'analisi più approfondita, verrà dato ad alcuni membri del gruppo il compito di valutarne gli impatti. In seguito, sarà nuovamente valutata con le informazioni aggiuntive durante una riunione successiva.

Implementazione della modifica

Una volta approvata una modifica, verrà creata per essa una issue su GitHub, poi tale modifica verrà implementata, verificata e rilasciata (come definito nella sezione 4.1.3).

Tracciabilità della modifica

La issue relativa alla modifica approvata, verrà riportata nel verbale della riunione in cui la modifica viene approvata. Tale issue, sarà composta come descritto nella sezione 4.1.2.

Grazie all'utilizzo di GitHub come Issue Tracking System sarà inoltre possibile risalire a ogni issue e all'autore della stessa.

3.2.4 Stato della configurazione

Per mantenere un controllo accurato, è necessario un metodo per identificare le varie versioni degli elementi software.

Versionamento

Ogni modifica provoca un aumento nella versione del documento riportato nel titolo dello stesso. Per informazioni sul versionamento si faccia riferimento alla sezione 3.1.2.

3.2.5 Valutazione della configurazione

La valutazione della configurazione sarà eseguita attraverso una serie di attività di verifica e validazione, tra cui revisioni del codice e test. L'obiettivo principale è garantire che il software sia conforme ai requisiti stabiliti, al fine di garantire la qualità e l'affidabilità del sistema.

3.2.6 Controllo del rilascio

Durante l'intero ciclo di vita del prodotto software, saranno conservate copie principali del codice e della documentazione. Questo processo assicurerà un'adeguata gestione e distribuzione dei rilasci software e della relativa documentazione. Le copie principali del codice e della documentazione saranno mantenute per consentire un tracciamento completo e un ripristino in caso di necessità.

3.3 Processo di controllo della qualità

Il processo di controllo della qualità ha l'obiettivo di garantire che i prodotti e i processi siano conformi ai requisiti.

3.3.1 Implementazione del processo

Per il controllo della qualità viene utilizzato il PDCA, descritto alla sezione 3.3.2, per favorire il miglioramento continuo della qualità dei prodotti e dei processi. Il processo di controllo della qualità sarà inoltre coordinato con i processi correlati di Verifica (sezione 3.4) e di Validazione (sezione 3.5).

3.3.2 PDCA

Il PDCA è un modello di gestione della qualità iterativo e ciclico utilizzato per il miglioramento continuo di processi. Esso è composto da quattro fasi:

1. **Plan**: fase in cui si pianificano le attività da svolgere per raggiungere un determinato obiettivo;
2. **Do**: fase in cui vengono eseguite le fasi pianificate durante la precedente fase (Plan);

3. **Check**: fase in cui vengono valutati i risultati ottenuti durante l'implementazione delle azioni (fase Do). Vengono confrontati i risultati con gli obiettivi pianificati nella fase di Plan;

4. **Act**: fase in cui vengono consolidate le cose positive emerse durante la fase precedente (Check). Se al contrario emergono problemi, verranno intraprese azioni correttive per raggiungere gli obiettivi desiderati. Grazie a questo l'esperienza guadagnata ad ogni iterazione permetterà di ottenere risultati migliori all'iterazione successiva.

3.3.3 Controllo del processo e del prodotto

Il controllo del processo avviene tramite la conformità agli standard, la verifica e la validazione della documentazione e la verifica della conformità del prodotto rispetto ai requisiti.

3.3.4 Strumenti per il controllo

Vengono utilizzate delle metriche per il controllo del processo e il controllo del prodotto, le quali sono riportate nell'appendice B.

Struttura delle metriche

Tali metriche vengono definite come descritto nella sezione B

Valori obiettivo delle metriche

I valori che le metriche devono assumere per essere ritenute accettabili o pienamente soddisfatte si trovano alla sezione B sotto il punto "descrizione" di ciascuna metrica.

3.4 Processo di verifica

Il processo di verifica serve per determinare se i prodotti software di un'attività soddisfano i requisiti o le condizioni imposte su di essi nelle attività precedenti. La verifica viene eseguita durante tutto il ciclo di vita del software per garantire la qualità, ridurre i costi, migliorare l'efficienza e garantire la soddisfazione del cliente.

3.4.1 Implementazione del processo

Questo processo viene svolto dai verificatori, i quali analizzano i prodotti per valutarne le conformità con quanto specificato nel documento *Piano di Qualifica v2.0.0.* Le attività di verifica vengono documentate nel documento *Piano di Qualifica v2.0.0*, descrivendone gli scopi, i risultati sperati e quelli ottenuti. Di seguito vengono descritte le attività che possono essere adottate.

3.4.2 Analisi statica

L'analisi statica è una tecnica utilizzata per valutare il codice e la documentazione che non richiede l'esecuzione del prodotto. Per questo tipo di analisi esistono due tecniche fondamentali: *walkthrough*$_G$ e *ispezione*$_G$ descritte qui di seguito:

Walkthrough

Il walkthrough è una pratica di revisione dei documenti e del codice che prevede una collaborazione tra verificatori e autori del prodotto; si articola in:

1. **pianificazione**: dialogo tra le due parti che ha lo scopo di identificare i vincoli che il prodotto deve soddisfare;
2. **lettura**: fase in cui i verificatori esaminano codice o documenti con l'obiettivo di individuare errori o non conformità rispetto ai vincoli stabiliti;
3. **discussione**: discussione tra autori e verificatori in cui i verificatori presentano eventuali punti critici individuati durante la lettura;
4. **correzione**: gli autori correggono gli errori individuati, garantendo così che il prodotto rispetti i requisiti stabiliti.

Questo approccio collaborativo e strutturato consente di individuare e correggere tempestivamente eventuali problemi nella documentazione o nel codice, contribuendo così a migliorare la qualità complessiva del prodotto software. Tuttavia, una debolezza di questa tecnica è il fatto di non essere automatizzabile, oltre alla sua informale natura di discussione. Tuttavia, per le prime fasi del progetto, non è possibile agire diversamente; pertanto i membri del gruppo utilizzeranno questa tecnica, la quale verrà documentata tramite i commenti nelle pull request sulla piattaforma GitHub.

3.4.3 Analisi dinamica

L'analisi dinamica è una tecnica utilizzata per valutare il comportamento di un programma in esecuzione (dunque non si applica per i documenti). Questo avviene tramite l'esecuzione di test. Esistono vari tipi di test, descritti qui di seguito.

Test di unità

Questa tipologia di test viene definita dai verificatori durante la fase di progettazione di dettaglio.
Questi test vengono definiti su singole unità di software testabili in maniera individuale. In particolare, se vengono eseguiti più test sulla stessa unità, questi formeranno una "test suite" per quella determinata unità.
Questa tipologia di test è a sua volta divisa in altre due tipologie di seguito descritte:

Tipo	Descrizione	Vantaggi	Svantaggi
Test funzionali (black box)	Comparano, dato un input, l'output effettivo con quello atteso. Producono "requirement-coverage"	Velocità di realizzazione	Non assicurano la copertura completa del flusso di esecuzione
Test strutturali (white box)	Vengono realizzati per eseguire ciascuno un flusso di esecuzione diverso. Una batteria di test assicura la copertura completa del codice in esame. Producono "structural-coverage"	Ogni flusso di esecuzione viene eseguito e valutato	In caso di complessità ciclomatica elevata, l'implementazione risulta complessa e probabile fonte di errori

Tabella 2: Tipologie di test di unità

Test di integrazione

I test di integrazione vengono definiti dai verificatori durante la fase di progettazione

architetturale e si concentrano sulla verifica della corretta interazione e integrazione tra diverse unità o componenti di un sistema. L'integrazione può avvenire con approccio *top-down*$_G$ o *bottom-up*$_G$.

Tipo	Descrizione	Vantaggi	Svantaggi
top-down	Le unità superiori del sistema vengono integrate e testate prima delle unità inferiori	Identifica problemi di alto livello in una fase precoce del processo di sviluppo	Probabili difficoltà nell'integrare le dipendenze tra unità superiori e inferiori del sistema, risultano necessari molti stub
bottom-up	Le unità inferiori del sistema vengono integrate e testate prima delle unità superiori	Identifica problemi di implementazione e codifica in una fase precoce del processo di sviluppo	Probabili ritardi nei test riguardanti l'architettura generale

Tabella 3: Tipologie di test di integrazione

Test di sistema

I test di sistema vengono definiti dai verificatori durante l'attività di analisi dei requisiti e vengono effettuati in seguito ai test di integrazione. Questi test sono progettati per verificare che l'intero sistema soddisfi i requisiti concordati. Sono essenziali per garantire che il sistema software sia pronto per l'utilizzo del cliente e che soddisfi le sue esigenze e aspettative.

Test di regressione

I test di regressione sono progettati per garantire che le modifiche al software non abbiano compromesso la funzionalità già testata e funzionante. In pratica, si tratta di ripetere i test di unità, integrazione e sistema per assicurarsi che le modifiche non intacchino funzionalità già verificate, causando così una regressione.

Test di accettazione

I test di accettazione accertano il soddisfacimento dei requisiti utente, con la condizione che devono essere svolti in presenza del committente.

Codici identificativi dei test

Ogni test è associato ad un codice identificativo così composto:

`T[tipo] - [codice]`

dove:

- `[T]`: è il tipo di test:
  - `[U]`: per i test di unità;
  - `[I]`: per i test di integrazione;
  - `[S]`: per i test di sistema;
  - `[A]`: per i test di accettazione.
- `[codice]`: codice identificativo del test tra tutti i test di quel tipo:
  - se il test non ha un padre, allora è un semplice numero progressivo;
  - se il test ha un padre, allora è nel formato:

    `[codice padre].[codice figlio]`

    dove:
    * `[codice_padre]`: è il codice che identifica il padre all'interno del suo tipo;
    * `[codice_figlio]`: è un numero progressivo tra i figli dello stesso padre.

Stato dei test

Ogni test ha uno stato che ne descrive il risultato:

- **NI**: il test non è ancora stato implementato;
- **Passato**: il test riporta esito positivo;
- **Non passato**: il test riporta esito negativo.

3.5 Processo di validazione

Il processo di validazione riguarda le attività coinvolte nel determinare se il prodotto finale soddisfi i requisiti e i vincoli previsti.

3.5.1 Implementazione del processo

Il processo di validazione prevede che, in seguito al superamento dei test durante le attività di verifica (descritta nella sezione 3.4), verrà effettuato il test di accettazione del progetto (3.5.1) che porterà alla validazione del progetto e, in caso di esito positivo, alla chiusura del progetto.

Test di accettazione del progetto

Il test di accettazione coincide con il collaudo ed è una dimostrazione in presenza del committente che il prodotto rispetti quanto richiesto nel capitolato d'appalto. Prima di procedere con il collaudo, i verificatori eseguono la test suite di sistema in un ambiente che replica quello di installazione. È necessario che i test di sistema diano esito positivo prima di procedere con il collaudo.

4 Processi organizzativi

I processi organizzativi indicati dallo standard ISO/IEC 12207:1997 servono per la gestione e lo sviluppo dell'organizzazione. Le attività e i compiti in ciascuno di questi processi sono affidati all'organizzazione stessa, che ne supervisiona e garantisce l'efficacia. L'obiettivo principale è garantire un'efficace gestione complessiva delle attività.

4.1 Gestione organizzativa

In questa sezione vengono esposte le norme relative l'organizzazione e il coordinamento delle attività interne ed esterne del gruppo, l'assegnazione dei ruoli e relativi compiti. Come stabilito dallo standard ISO/IEC 12207:1997 il processo di gestione identifica le seguenti attività:

1. inizializzazione e definizione dello scopo 4.1.1;
2. pianificazione 4.1.2;
3. esecuzione e controllo 4.1.3;
4. revisione e valutazione 4.1.4.

4.1.1 Inizializzazione e definizione dello scopo

Questo processo ha i seguenti obiettivi:

- pianificare con criterio le attività;
- monitorare in modo efficace il gruppo, le risorse e i processi;
- assegnare correttamente ruoli e compiti;
- facilitare la comunicazione interna ed esterna.

Il responsabile di progetto si impegna a:

- assegnare i ruoli e i relativi compiti ai membri del gruppo;
- amministrare gli strumenti di coordinamento concordati;
- stimare i rischi;

- gestire gli imprevisti;
- gestire le comunicazioni interne ed esterne;
- calcolare i preventivi relativi alle ore e i costi divisi per ruolo;
- calcolare i consuntivi di ogni periodo;
- determinare quando un processo, attività o task è giunto al termine e approvarlo.

4.1.2 Pianificazione

Nell'attività di pianificazione il responsabile prepara i piani dettagliati per l'esecuzione delle attività. In particolare verranno riportati:

- una descrizione completa dei compiti;
- la definizione dei tempi di completamento;
- l'assegnazione delle risorse;
- la valutazione dei rischi e relativa implementazione di misure di controllo;
- un'analisi dei costi.

Questi piani verranno redatti dal responsabile e si troveranno all'interno del documento `Piano di Progetto`.

Questa sezione relativa alla pianificazione sarà strutturata come segue:

- gestione Backlog;
- gestione Sprint;
- ruoli;
- valutazione rischi;
- preventivi.

Gestione Backlog

Per la gestione delle attività è necessario rifarsi alla piattaforma Notion. Qui è presente un database contenente il Backlog completo. In particolare, sono riportati, per ogni issue:

- **titolo**: indica il nome della issue;
- **stato**: indica lo stato della issue. Ogni issue può avere soltanto uno tra i seguenti stati:
  - to-do: compito non ancora iniziato. Si noti che al momento dell'inserimento nel database, una issue ha sempre lo stato settato a to-do;
  - in-progress: compito in corso;
  - done: compito completato;
- **importanza**: indica il grado di importanza associato;
- **sprint**: indica lo Sprint in cui la issue è assegnata.

Sono inoltre presenti delle viste al database per semplificarne la visione e avere un'idea più chiara sull'andamento del progetto. Le viste presenti sono:

- **board-generica**: mostra una board generale in cui tutte le issue sono divise tra to-do, in-progress e done;
- **board-sprint**: mostra una board in cui sono presentate solo le issue relative allo Sprint in corso divise tra to-do, in-progress e done.

Gestione Sprint

All'inizio di ogni Sprint, durante la fase di pianificazione, si procede a modificare o impostare il valore del campo "sprint" all'interno del database descritto nella sezione 4.1.2. Questo si rende necessario poiché alcuni compiti vengono pianificati inizialmente, mentre altri vengono definiti successivamente.
Durante lo Sprint, è importante notare che anche in un momento successivo alla pianificazione alcune attività possono essere aggiunte nonostante non siano state originariamente pianificate. Questo avviene in base alle competenze possedute dal team e attraverso il riferimento al Backlog generale. Questa pratica permette di massimizzare il valore del lavoro svolto e mantenere una certa flessibilità.

Dopo aver selezionato quali compiti faranno parte dello Sprint, il responsabile si occuperà di creare su GitHub una milestone relativa allo Sprint e le relative issue (descritte nella sezione 4.1.2) a cui verranno associati i vari membri del gruppo a seconda dei loro ruoli per quel determinato Sprint, descritti alla sezione 4.1.2.
Questa pratica permette di associare in modo chiaro i compiti allo Sprint in corso, consentendo al team di avere una visione chiara del lavoro e di monitorarne il progresso in maniera efficace.

Creazione issue

Il Responsabile si occupa di creare le issue in seguito alla pianificazione. Ciascuna issue sarà così composta:

- **titolo**: titolo descrittivo della issue;
- **assignees**: membro assegnato alla issue;
- **descrizione**: codice di riferimento al verbale interno, così composto:

  `VI-[data_verbale]-#[numero_issue]`

  dove:

  - `[data_verbale]` corrisponde alla data del verbale in cui è nata la issue, così formata: aaaa-mm-gg;
  - `[numero_issue]` corrisponde al numero della issue su GitHub.
- **labels**: labels come descritte nella sezione 3.1.2;
- **projects**: il progetto presente su GitHub denominato "Dashboard avanzamento progetto";
- **milestone**: milestone di riferimento allo Sprint.

Ruoli

Per garantire una suddivisione efficace delle attività, saranno delineati sei ruoli che i membri del team assumeranno durante lo svolgimento del progetto. I ruoli verranno assegnati all'inizio di ogni Sprint. Ogni membro del team è tenuto a coprire almeno una volta ognuno dei ruoli, che vengono elencati qui di seguito:

- **Responsabile**: svolge la funzione di coordinatore del gruppo e di rappresentante del team nei confronti di committente e proponente per tutta la durata del progetto. Ha molteplici responsabilità:
  - predisposizione e gestione delle risorse;
  - coordinamento dei membri del team;
  - verifica dello stato di attività e processi;
  - relazioni con le figure esterne al team;
  - approvazione dei documenti di progetto.
- **Amministratore**: definisce, controlla e amministra l'ambiente di lavoro e le risorse messe a disposizione per tutto il periodo di sviluppo del progetto. Deve accertarsi che i mezzi messi a disposizione perseguano produttività, garantendo allo stesso tempo qualità ed economicità.
  Redige le `Norme di Progetto` ed è sua responsabilità verificare che i membri del team le seguano.
  È inoltre sua responsabilità:
  - amministrare infrastrutture, strumenti e documentazione;
  - gestione degli imprevisti legati alla gestione dei processi;
  - redigere e mantenere aggiornata la documentazione e il versionamento della stessa;
  - gestire la configurazione del prodotto.
- **Analista**: è una figura molto competente, a cui ci si affida per scomporre e approfondire le esigenze del committente. Il suo ruolo è fondamentale nelle prime fasi del progetto, in particolare per la stesura del file `Analisi dei Requisiti` che sarà una serie di specifiche tecniche che saranno fondamentali per le fasi successive di sviluppo.
  Si occupa di:
  - mediare fra proponenti/committenti e sviluppatori;
  - studia le necessità dei proponenti definendo problemi, obiettivi e requisiti soluzione;
  - etichetta i requisiti in: impliciti/espliciti e opzionali/obbligatori;

  - redige i file `Studio di Fattibilità` e `Analisi dei Requisiti`.

- **Progettista**: ha l'onere di sviluppare una soluzione che soddisfi in maniera accettabile i vincoli dati dall'analista. Effettua scelte tecniche e tecnologiche, segue lo sviluppo, non la manutenzione.
  Si occupa di:

  - sviluppare un'architettura robusta seguendo le best practises perseguendo coerenza e consistenza;
  - ricercare soluzioni efficienti ed efficaci che soddisfino i requisiti nel rispetto dei vincoli dati;
  - decomporre il sistema in componenti e organizzarne le interazioni fra di essi;
  - decomporre ruoli e responsabilità in favore di modularizzazione e riutilizzo;
  - usare soluzioni ottimizzate.

- **Programmatore**: ha competenze tecniche e appartiene alla categoria più popolosa del gruppo. Si occupa di:

  - codificare la soluzione in modo mantenibile;
  - partecipare a implementazione e manutenzione del prodotto;
  - creare test ad hoc per la verifica e valutazione del codice;
  - redige il manuale utente.

- **Verificatore**: il suo compito è quello di controllare il lavoro svolto dagli altri membri del gruppo.
  Deve assicurarsi che:

  - i compiti vengano svolti come da programma, altrimenti offre spunti per le correzioni;
  - l'esecuzione delle attività di processo non causi errori.

Valutazione rischi

Il responsabile si occupa di riconoscere i possibili rischi e registrarli nel documento `Piano di Progetto`. Una volta individuati, è fondamentale delineare una o più strategie mirate alla gestione di tali rischi.

I rischi vengono categorizzati in base alla natura del problema potenziale. Le tipologie individuate sono:

- rischi tecnologici;
- rischi legati alle persone;
- rischi organizzativi;
- rischi sulle stime;
- rischi sui requisiti.

Ogni rischio è caratterizzato da:

- intestazione: i rischi vengono identificati mediante codici univoci creati appositamente per questo scopo e un nome, in questo formato:

  `R[categoria][numero]-[nome]`

  dove:

  - `[categoria]` corrisponde alla tipologia del rischio:
    - `T` se è un rischio tecnologico;
    - `P` se è un rischio legato alle persone;
    - `O` se è un rischio organizzativo;
    - `S` se è un rischio sulle stime;
    - `R` se è un rischio sui requisiti;
  - `[numero]`: è un numero progressivo per quella categoria di rischio;
  - `[nome]`: corrisponde al nome dato al rischio.
- descrizione e conseguenze;
- valutazione della probabilità di occorrenza e della pericolosità;
- piano di controllo;
- piano di contingenza.

4.1.3 Esecuzione e controllo

L'attività di esecuzione e controllo delinea le pratiche necessarie a guidare l'attuazione del piano per raggiungere gli obiettivi, monitorare attentamente il processo e fornire report sia interni che esterni. Si definisce inoltre la gestione dei problemi emergenti, includendo l'analisi, la risoluzione e l'eventuale adattamento dei piani.
Questa sezione relativa all'esecuzione e il controllo sarà strutturata come segue:

- creazione issue;
- gestione issue;
- comunicazioni interne;
- comunicazioni esterne;
- incontri interni;
- incontri esterni;
- verbali;
- controllo rischi;

Gestione issue

Nel momento in cui ad un membro del gruppo viene assegnata una issue, il procedimento è il seguente:

1. **creazione del nuovo branch**: il membro a cui viene assegnata la issue crea un nuovo branch a partire da `develop` usando la sezione *Development* nel form della issue su GitHub;
2. **sviluppo delle attività**: vengono sviluppate le attività richieste; ogni volta che una parte del lavoro è completata, le modifiche vengono pubblicate sul proprio branch, in modo da renderle disponibili a tutto il team di QB Software;
3. **aggiornamento registro e pull request**: quando la task è terminata, si aggiorna il registro delle modifiche e si richiede una pull request dal proprio branch verso il `develop`;

4. **verifica**: il verificatore si auto-assegna e dovrà verificare i contenuti secondo quanto riportato nella sezione 4.1.4;

5. **approvazione o correzione delle modifiche**: il verificatore, se approva il contenuto, deve fare il merge della pull request, integrando le modifiche nel branch di sviluppo principale e eliminando il branch creato per la issue. In caso di rifiuto, il membro del team a cui era assegnata la issue deve assolvere alle mancanze e iniziare nuovamente la procedura di verifica.

Comunicazioni interne

Le comunicazioni interne possono avvenire tramite:

- **Whatsapp**: servizio di messaggistica istantanea utilizzato dai membri del team per comunicazioni rapide e informali;
- **Discord**: servizio di messaggistica e videochiamate utilizzato dal team per le riunioni interne da remoto. Offre inoltre la possibilità di creare numerosi canali di comunicazione, questo risulta molto utile per strutturare la comunicazione dividendola per argomenti.

Comunicazioni esterne

Le comunicazioni esterne sono gestite dal Responsabile e possono avvenire tramite:

- **posta elettronica**: l'indirizzo di posta elettronica del gruppo è

  qbsoftware.swe@gmail.com

- **Discord**: viene creato un apposito canale di comunicazione condiviso con il proponente per questioni rapide.

Riunioni interne

Le riunioni interne del team avvengono interamente online tramite la piattaforma Discord. La data e l'ora vengono decise tramite i servizi descritti alla sezione 4.1.3. In seguito sarà compito del responsabile creare un evento relativo all'incontro previsto su Google Calendar dell'account del gruppo per permettere a tutto il gruppo di visionarli. Le riunioni interne generano un verbale, come descritto nella sezione 4.1.3.

Riunioni esterne

Le riunioni esterne prevedono la partecipazione di almeno 5 dei membri del gruppo. Le riunioni esterne si terranno su Carbonio Chats system: piattaforma per videochiamate sviluppata dal proponente.
Le riunioni esterne generano un verbale, come descritto nella sezione 4.1.3.

Verbali

Le comunicazioni esterne e tutte le riunioni, sia interne che esterne, generano un verbale. Durante le riunioni, il responsabile designa un membro del gruppo incaricato di prendere appunti su Notion. Questi appunti sono disponibili per tutto il gruppo e saranno utilizzati per redigere il verbale.
Il membro del gruppo incaricato della stesura del verbale viene scelto dal Responsabile.

Controllo rischi

È compito del responsabile assicurarsi che, nel caso di attualizzazione di un rischio, il relativo piano di contingenza previsto venga seguito. È inoltre compito del responsabile aggiornare, ogni qual volta si renda necessario, i rischi e i relativi piani di contingenza.
Nella fase di retrospettiva di ciascuno Sprint, descritta nella sezione 4.1.4, verranno discussi i rischi incontrati.

4.1.4 Revisione e valutazione

Questa attività coinvolge la valutazione dei prodotti e dei piani d'azione per adempiere ai vincoli contrattuali, nonché la valutazione delle attività e dei prodotti software completati durante l'esecuzione del processo per il raggiungimento degli obiettivi pianificati.
Questa sezione relativa alla revisione e alla valutazione sarà strutturata come segue:

- retrospettiva;
- verifica.

Retrospettiva

Alla fine di ogni Sprint, verrà discusso con il team quanto fatto in una riunione, in particolare il responsabile creerà un file Notion con i seguenti punti da affrontare:

- cose che hanno funzionato;

- cose che non hanno funzionato;
- rischi incontrati.

Questo documento verrà utilizzato come base di appunti per il verbale relativo alla riunione. Successivamente, il Responsabile riporterà un rapido riassunto di quanto riportato nel verbale anche nella sezione Consuntivi nel file Piano di Progetto, in particolare per i rischi incontrati durante lo Sprint e una valutazione del piano di mitigazione previsto.

Gestione oraria

Per la gestione delle ore il team utilizza un Google Sheet condiviso sull'account aziendale. Questa pratica permette di gestire in modo trasparente il tempo dedicato alle attività e facilità la valutazione della produttività.
Alla fine di ogni Sprint, il responsabile redigerà poi un consuntivo nel documento Piano di Progetto, includendo i costi prodotti dalle ore segnate nel file in questione.

Verifica

La verifica viene fatta attraverso il metodo del *walkthrough*, cioè il verificatore legge l'intero documento o la rispettiva porzione di codice e controlla la soddisfazione delle modifiche richieste dalla issue.
La verifica per la fase di manutenzione avviene attraverso delle *checklist* presenti nel `Piano di Qualifica`. Terminata la procedura di verifica il verificatore deve pubblicare un commento sulla pull request riportando il feedback della verifica.

4.2 Processo di miglioramento

Il processo di miglioramento è un processo che ha lo scopo di controllare, misurare e migliorare i diversi processi. Questo processo è composto dalle seguenti attività:

1. implementazione dei processi 4.2.1;
2. valutazione dei processi 4.2.2;
3. miglioramento dei processi 4.2.3.

4.2.1 Implementazione dei processi

I processi devono essere stabiliti a partire dallo standard ISO/IEC 12207:1997 per poi essere implementati nel way of working del team.

4.2.2 Valutazione dei processi

I processi devono essere valutati per verificare che siano stati implementati correttamente e che siano in grado di soddisfare i requisiti richiesti. La valutazione deve essere un processo continuo, condotto in maniera tale da identificare e risolvere tempestivamente qualsiasi problema che possa emergere.

4.2.3 Miglioramento dei processi

A seguito della valutazione, i processi devono essere migliorati sfruttando dati storici e metriche per identificare i punti critici e le aree di miglioramento.

4.3 Processo di formazione

Il processo di formazione è un processo che ha lo scopo di formare i membri del team in modo da garantire che essi siano in grado di svolgere i compiti assegnati. Questo processo è composto dalle seguenti attività:

1. implementazione del processo di formazione (sezione 4.3.1);
2. individuazione e creazione di documentazione (sezione 4.3.1);
3. formazione dei membri del gruppo (sezione 4.3.2).

4.3.1 Implementazione del processo di formazione

Per formare i membri del team è necessario capire il dominio del problema che viene affrontato nel progetto e le tecnologie che verranno utilizzate per affrontarlo. I diversi strumenti usati nello sviluppo sono elencati nella sezione 3.1.2. Bisogna individuare la documentazione necessaria per formare i membri del team, questa documentazione viene reperita da fonti esterne. La quantità di documentazione continua a crescere man mano che il progetto procede. Nel caso in cui la documentazione esistente non sia sufficiente deve essere creata nuova documentazione per permettere ai membri del team di apprendere in modo autonomo le tecnologie che verranno utilizzate durante lo sviluppo del progetto. La documentazione deve essere facilmente comprensibile e accessibile a tutti i membri del team.

4.3.2 Formazione dei membri del gruppo

I membri del gruppo devono impegnarsi a formarsi autonomamente, utilizzando la documentazione fornita, in modo da poter affrontare il progetto. La formazione può avvenire sia in modo autonomo, sia in modo collaborativo, in modo da poter condividere le conoscenze acquisite.

A Standard per la qualità

Le norme per la qualità fanno riferimento allo Standard l'ISO/IEC 9126, che fornisce diverse caratteristiche per garantire la qualità del prodotto software. In tal modo, si assicura che il prodotto finale soddisfi i vincoli contrattuali e le esigenze del destinatario. Il modello è articolato nel seguente modo:

- **funzionalità**;
- **affidabilità**;
- **efficienza**;
- **usabilità**;
- **manuntenibilità**;
- **portabilità**.

Ogni categoria viene suddivisa in sottocategorie per fornire una granulosità di dettaglio maggiore per quanto concerne la qualità del software.

A.1 Funzionalità

È la caratteristica del prodotto software che fa riferimento alle sue prestazioni effettive, rispetto a quelle previste da contratto. Le caratteristiche che la compongono sono:

- **adeguatezza**: è la capacità del software di fornire una serie di funzioni che consentano di soddisfare adeguatamente le esigenze degli utenti finali;
- **accuratezza**: è la capacità del software di fornire risultati precisi e corretti, in linea con quanto preventivato;
- **interoperabilità**: è la capacità del software di interagire con altri sistemi;
- **sicurezza**: è la capacità del software di proteggere informazioni e dati, negando a chi non autorizzato, di accedervi;
- **conformità**: è la capacità del software di aderire agli standard, convenzioni e regolamentazioni del settore.

A.2 Affidabilità

È la caratteristica con cui si misura la capacità del software di mantenere un determinato livello di prestazioni. Le caratteristiche che la compongono sono:

- **maturità**: è la capacità del software di non produrre errori, malfunzionamenti e risultati non corretti;
- **tollerenza agli errori**: è la capacità del software di gestire in modo appropriato gli scenari in cui sono presenti errori e malfunzionamenti;
- **recuperabilità**: è la capacità del software di ristabilire un certo livello di prestazioni, a seguito di un malfunzionamento;
- **aderenza**: è la capacità del software di aderire agli standard, regole e convenzioni inerenti all'affidabilità.

A.3 Efficienza

È la caratteristica con cui si misura la capacità del software di avere determinati livelli di prestazioni, gestendo in maniera appropriata le risorse del sistema. Le sottocaratteristiche che la compongono sono:

- **comportamento rispetto al tempo**: è la capacità del software di fornire adeguati tempi di risposta, calcolo e quantità di lavoro;
- **utilizzo delle risorse**: è la capacità del software di utilizzare le propriè risorse in maniera adeguata;
- **conformità**: è la capacità del software di aderire alle convenzioni e agli standard dell'efficienza.

A.4 Usabilità

È la caratteristica con cui si misura la capacità del software di essere facilmente capibile e utilizzabile dall'utente finale. Le sottocaratteristiche che la compongono sono:

- **comprensibilità**: è la capacità del software di essere facilmente comprensibile, mettendo a proprio agio l'utente;

- **apprendibilità**: è la capacità del software di minimizzare l'impegno richiesto agli utenti per imparare ad usare il prodotto;
- **operabilità**: è la capacità del software di mettere a proprio agio l'utente per fargli usare il prodotto secondo i suoi scopi;
- **attrattiva**: è la capacità del software di essere gradevole all'utente;
- **conformità**: è la capacità del software di aderire alle convenzioni e agli standard dell'usabilità.

A.5 Manuntenibilità

È la caratteristica con cui si misura la capacità del software di essere modificato, con correzioni, miglioramenti e adattamenti. Le sottocaratteristiche che la compongono sono:

- **analizzabilità**: è la capacità del software di essere facilmente analizzabile, in modo da trovare bachi;
- **modificabilità**: è la capacità del software di poter accogliere facilmente nuove modifiche;
- **stabilità**: è la capacità del software di evitare comportamenti erronei a seguito di modifiche errate;
- **testabilità**: è la capacità del software di essere facilmente testato per validare i comportamenti apportati.

A.6 Portabilità

È la caratteristica con cui si misura la capacità del software di essere utilizzato e trasferito in ambienti di lavoro diversi. Le sottocaratteristiche che la compongono sono:

- **adattabilità**: è la capacità del software di essere adatto a diversi ambienti senza apportare modifiche al sistema;
- **installabilità**: è la capacità del software di essere installato in un certo ambiente;
- **conformità**: è la caratteristica del software di aderire alle convenzioni e agli standard della portabilità;

- **sostituibilità**: è la caratteristica del software di sostituire un altro prodotto software, con la stessa funzione nel medesimo ambiente.

B Metriche per il Controllo di Qualità

In questa sezione riportiamo tutte le metriche utilizzate nel progetto. Le metriche sono divisi in due macrogruppi:

- **Qualità di Processo**, dove vengono raccolte tutte le metriche che misurano i processi. Le metriche presenti in questo categoria ci permettono di capire quanto è buona la $governance_G$ del progetto;
- **Qualità di Prodotto**, dove vengono raccolte tutte le metriche che misurano la qualità dei prodotti;

Ogni metrica ha un **codice identificativo** (ID) che è composto da due parti, un prefisso che può essere:

- `MW`, dove 'MW' sta per *Metriche per la qualità del Way of working*, ovvero quelle metriche che appartengono al gruppo Qualità di Processo;
- `MP`, dove 'MP' sta per *Metriche per la qualità del Prodotto*;

e numero progressivo che enumera le varie metriche. Ogni metrica verrà introdotta con il seguente formato:

$$[ID] - [Nome\,completo\,della\,metrica]$$

e una lista che riporta le seguenti informazioni:

- **descrizione**, espone a cosa serve la metrica, e se necessario chiarisce l'interpretazione da dare al risultato;
- **obiettivo**, indica qual è lo scopo della metrica;
- **target**, solo per le metriche riguardanti la qualità di processo, specifica qual è il processo valutato attraverso la metrica;
- **dominio**, su quale intervallo la metrica assume valori ammissibili;
- **unità di misura**, quale unità usa la metrica per esprimere il risultato della formula;

- **formula**, come calcolare la metrica;
- **input**, gli indicatori caratteristici, oggettivi e quantitativi dell'oggetto preso in esame dalla metrica, che permettono di calcolare il valore della metrica attraverso la formula.

Alcune informazioni possono essere omesse, ad esempio alcune formule e i loro input quando l'algoritmo di calcolo è tanto complesso ed è possibile utilizzare un software per calcolare il valore della metrica.

B.1 Metriche per la Qualità di Processo

- **MW1-VP - Variazioni di Piano**
  - **descrizione**: calcola il rapporto tra i giorni che effettivamente sono stati necessari a finire lo Sprint rispetto a quelli pianificati. Dunque è possibile capire se c'è un ritardo o un anticipo rispetto a quanto pianificato per uno Sprint. Il valore va interpretato nel seguente modo:
    * se MW1-VP $> 0\%$ $\implies$ si è in anticipo rispetto a quanto pianificato, generalmente se questo valore è molto lontano dallo zero e ripetuto nel tempo allora assume una eccezione negativa, perché potrebbe significare che la pianificazione non ha effettivamente consapevolezza delle risorse messe a disposizione per lo Sprint e che il carico di lavoro per lo Sprint è stato sottostimato;
    * se MW1-VP $= 0\%$ $\implies$ si è perfettamente in linea con quanto pianificato, teoricamente è l'ideale;
    * se MW1-VP $< 0\%$ $\implies$ si è in ritardo rispetto a quanto si è pianificato, se presenta una valore molto basso o ripetuto nel tempo significa che c'è qualcosa di sbagliato nella pianificazione o nel modo di lavorare;
  - **obiettivo**: determinare se ci sono problemi e/o cambiamenti che rallentano o anticipano la pianificazione, il valore ottimale è un numero negativo ma comunque vicino allo zero;
  - **target**: processi primari $\rightarrow$ fornitura (sezione 2.1);
  - **dominio**: MW1-VP $\in \mathbb{Z}$;
  - **unità di misura**: percentuale;

  - **formula**: $\frac{(F_E - I_E)}{(F_S - I_S)}$;
  - **input**:
    * F_S: fine preventivato dello Sprint (data);
    * I_S: inizio preventivato dello Sprint (data);
    * F_E: fine effettivo dello Sprint (data);
    * I_E: inizio effettivo dello Sprint (data).

- **MW2-IVC - Indice Variazioni di Costo**
  - **descrizione**: misura le variazioni di costo rispetto a quanto preventivato per uno Sprint. Il valore va interpretato nel seguente modo:
    * se MW2-IVC $> 1 \implies$ si è speso più di quanto preventivato per lo Sprint;
    * se MW2-IVC $= 1 \implies$ si è speso il tutto il budget preventivato per lo Sprint;
    * se MW2-IVC $< 1 \implies$ non è stato speso tutto il budget preventivato per lo Sprint;
  - **obiettivo**: determinare se il progetto è in linea con i costi stimati durante la pianificazione dello Sprint, e ridurre al minimo le spese mantenendo la qualità dei processi e/o prodotti. Questo metrica deve essere minimizzata;
  - **target**: processi primari $\rightarrow$ fornitura (sezione 2.1);
  - **dominio**: MW2-IVC $\in \mathbb{R}_+$;
  - **unità di misura**: percentuale;
  - **formula**: $\frac{C_E}{C_P}$;
  - **input**:
    * C_P: costo preventivato durante la pianificazione dello Sprint (euro);
    * C_E: costo effettivamente speso durante lo Sprint (euro).

- **MW3-VR - Variazioni dei Requisiti**
  - **descrizione**: misura la variazione dei requisiti e dunque la stabilità dell'analisi dei requisiti nel tempo. Quando per un certo intervallo di tempo questa metrica presenta valori molto distanti dallo zero significa che le l'analisi dei

requisiti non è stabile perché continua a cambiare. Quando la metrica assume come valore zero per un certo intervallo di tempo allora significa che l'analisi dei requisiti è stabile;

  - **obiettivo**: determinare la stabilità dei requisiti fin'ora raccolti. Quando si è all'inizio nel sviluppare l'analisi dei requisti il valore ottimo è positivo, ma man mano che ci si avvicina alla RTB in poi il valore ottimo è zero;
  - **target**: processi primari $\rightarrow$ sviluppo (sezione 2.2);
  - **dominio**: MW3-VR $\in \mathbb{Z}_{+}$;
  - **unità di misura**: unità;
  - **formula**: $R_{+} + R_{-} + R_{\#}$;
  - **input**:
    * R_{+}: requisiti aggiunti (numero naturale);
    * R_{-}: requisiti rimossi (numero naturale);
    * $R_{\#}$: requisiti modificati (numero naturale).

- **MW4-PMS - Percentuale di Metriche Soddisfatte**:
  - **descrizione**: misura il numero di metriche che soddisfano i requisti minimi imposti. Questa metrica permette di avere un panoramica generale sullo stato di salute del Sistema di Qualità, e dunque anche lo stato di salute del modo di lavorare e del prodotto;
  - **obiettivo**: determinare la qualità delle metriche scelte e se esiste qualcosa che non permette al modo scelto da QB Software di valutare e applicare un costante miglioramento nel modo corretto. Il valore ottimo è 100%;
  - **target**: processi di supporto $\rightarrow$ controllo della qualità (sezione 3.3);
  - **dominio**: MW4-PMS $\in [0, 100]$;
  - **unità di misura**: percentuale;
  - **formula**: $\frac{M_s}{M_t}$;
  - **input**:
    * M_s: numero di metriche soddisfatte (numero naturale);
    * M_t: numero di metriche totali (numero naturale diverso da zero).

- **MP5-PRNPI - Percentuale di Rischi Non Preventivati tra i rischi Incontrati**
  - **descrizione**: misura il numero di rischi non preventivati per uno Sprint rispetto a quelli che sono stati già preventivati. Se la metrica assume valore 0% vuol dire che per lo Sprint sono stati incontrati solo rischi già preventivati e dunque esiste già una risposta, se la metrica è maggiore dello zero percento significa che è stato incontrato qualche rischio non preventivato;
  - **obiettivo**: valutare se i rischi preventivati sono sufficientemente buoni per coprire la maggior parte delle problematiche che si può incontrare durante il progetto. Il valore ottimo è 0%;
  - **target**: processi organizzativi $\rightarrow$ gestione organizzativa (sezione 4.1);
  - **dominio**: MP5-PRNPI $\in [0, 100]$;
  - **unità di misura**: percentuale;
  - **formula**: $\frac{R_{NPI}}{R_{NPI}+R_{PI}}$, se non sono stati incontrati rischi la metrica va messa a 0%;
  - **input**:
    * R_{NPI}: numero di rischi non preventivati incontrati nello Sprint (numero naturale);
    * R_{PI}: numero di rischi preventivati incontrati nello Sprint (numero naturale).
- **MW6-IG - Indice di GULPEASE**
  - **descrizione**: misura la leggibilità di un documento;
  - **obiettivo**: rendere il più semplice possibile la lettura del documento per garantire una facile comprensione a tutto il gruppo e riduce la possibilità di equivoci dovuti alla complessità del documento;
  - **target**: processi di supporto $\rightarrow$ documentazione (sezione 3.1);
  - **dominio**: MW6-IG $\in [0, 100]$;
  - **unità di misura**: unità;
  - **formula**: $89 + \frac{300 \cdot N_F - 10 \cdot N_L}{N_P}$;
  - **input**:

    * N_F: numero delle frasi (numero naturale);
    * N_L: numero medio delle lettere presenti in una frase (numero naturale);
    * N_P: numero medio di parole presenti in una frase (numero naturale diverso da zero).

- **MW7-NEO - Numero di Errori Ortografici**
  - **descrizione**: misura la presenza di errori ortografici all'interno di un documento;
  - **obiettivo**: rendere più chiari i documenti. Il valore ottimo è zero errori ortografici;
  - **target**: processi di supporto $\rightarrow$ documentazione (sezione 3.1);
  - **dominio**: MW7-NEO $\in \mathbb{N}$;
  - **unità di misura**: errori;
  - **formula**: E;
  - **input**:
    * E: numero di errori ortografici all'interno del documento (numero naturale).

- **MW8-MOPOO - Media Ore Produttive vs Ore di Orologio**
  - **descrizione**: misura il rapporto tra il tempo effettivamente richiesto (ore di orologio) e il tempo stimato (ore produttive) per eseguire un processo. Se $R < 1$ allora il gruppo guadagna un margine utile, mentre se $R > 1$ allora c'è almeno una persona che ha difficoltà a portare al termine la task rispetto ai tempi previsti;
  - **obiettivo**: valutare la pressione esercitata sul gruppo. Capire quanto sappiamo fare e quanto capaci siamo nel svolgere le attività assegnate;
  - **target**: generale, la misura vale per tutti i processi;
  - **dominio**: MW8-MOPOO $\in \mathbb{R}_+$;
  - **unità di misura**: unità;
  - **formula**: $\frac{O_O}{O_P}$;
  - **input**:

    * O_O: ore richieste di orologio per eseguire la task assegnata (numero reale positivo);
    * O_P: ore preventivate per la produzione di quanto richiesto (numero reale positivo diverso da zero).

- **MW9-DTC - Debito Tecnico sul Codice**
  - **descrizione**: misura il debito tecnico rispetto allo sviluppo delle varie parti, contando i TODO e i FIXME presenti nel codice;
  - **obiettivo**: valutare il debito tecnico a livello di codifica, il valore preferibile sarebbe 0;
  - **target**: sviluppo;
  - **dominio**: MW9-DTC $\in \mathbb{R}_+$;
  - **unità di misura**: unità;
  - **formula**: N_{FT};
  - **input**:
    * N_{FT}: numero di FIXME/TODO presenti nel codice.
- **MW10-RCTC - Rapporto Commenti sul Totale del Codice**
  - **descrizione**: misura il rapporto tra le linee di commenti e il numero totale di righe di codice. Permette di capire a grandi linee se il codice è comprensibile e non necessità di commenti per spiegare il suo funzionamento o cosa fanno certe parti del programma;
  - **obiettivo**: valutare la comprensibilità del codice, un valore accettabili è al disotto del 3%;
  - **target**: sviluppo;
  - **dominio**: MW10-RCTC $\in \mathbb{R}_+$;
  - **unità di misura**: unità;
  - **formula**: $\frac{N_C}{N_T}$;
  - **input**:
    * N_C: numero di linee commento (numero naturale);
    * N_T: numero di linee di codice totali (numero naturale diverso da zero).

B.2 Metriche per la Qualità di Prodotto

B.2.1 Funzionalità

- **MP1-PROS - Percentuale dei Requisiti Obbligatori Soddisfatti**
  - **descrizione**: misura il rapporto di requisti obbligatori implementati rispetto al totale dei requisiti obbligatori;
  - **obiettivo**: valutare lo stato dell'implementazione dei requisiti obbligatori richiesti. Il valore che deve essere raggiunto è 100%;
  - **dominio**: $MP1 - PROS \in [0, 100]$;
  - **unità di misura**: percentuale;
  - **formula**: $\frac{R_{OS}}{R_O}$;
  - **input**:
    * R_{OS}: numero di requisiti obbligatori soddisfatti (numero naturale);
    * R_O: numero di requisiti obbligatori (numero naturale diverso da zero).
- **MP2-PRDS - Percentuale dei Requisiti Desiderabili Soddisfatti**
  - **descrizione**: misura il rapporto di requisti desiderabili implementati rispetto al totale dei requisiti desiderabili;
  - **obiettivo**: valutare lo stato dell'implementazione dei requisiti desiderabili richiesti;
  - **dominio**: $MP2 - PRDS \in [0, 100]$;
  - **unità di misura**: percentuale;
  - **formula**: $\frac{R_{DS}}{R_D}$;
  - **input**:
    * R_{DS}: numero di requisiti desiderabili soddisfatti (numero naturale);
    * R_D: numero di requisiti desiderabili (numero naturale diverso da zero).
- **MP3-PRFS - Percentuale dei Requisiti Facoltativi Soddisfatti**
  - **descrizione**: misura il rapporto di requisti facoltativi implementati rispetto al totale dei requisiti facoltativi;

- **obiettivo**: valutare lo stato dell'implementazione dei requisiti facoltativi richiesti;
- **dominio**: $MP3 - PRFS \in [0, 100]$;
- **unità di misura**: percentuale;
- **formula**: $\frac{R_{FS}}{R_F}$;
- **input**:
  * R_{FS}: numero di requisiti facoltativi soddisfatti (numero naturale);
  * R_F: numero di requisiti facoltativi (numero naturale diverso da zero).

B.2.2 Affidabilità

- **MP4-SC - Statement Coverage**
  * **descrizione**: misura il rapporto di comandi eseguiti correttamente dai test rispetto al totale dei comandi;
  * **obiettivo**: avere maggiori garanzie che le unità implementate funzioni nel modo atteso. Il valore ottimale è 100%;
  * **dominio**: MP4-SC $\in [0, 100]$;
  * **unità di misura**: percentuale;
  * **formula**: $\frac{S_{SC}}{S_T}$;
  * **input**:
    · S_{SC}: numero di comandi coperti da almeno un test (numero naturale);
    · S_T: numero di comandi totali presenti (numero natural positivo diverso da zero).
- **MP5-BC - Branch Coverage**
  * **descrizione**: misura il numero di rami decisionali che sono coperti dai test;
  * **obiettivo**: valutare che tutto il prodotto venga effettivamente testato. Il valore ottimale è 100%;
  * **dominio**: MP5-BC $\in [0, 100]$;
  * **unità di misura**: percentuale;

    * **formula**: calcolata con un plugin di Maven.

- **MPP6-CD - Code Duplication**
    * **descrizione**: misura la duplicazione del codice all'interno del prodotto;
    * **obiettivo**: valuta che non ci siano una duplicazione del codice molto alta che porta ad una pessima manutenibilità, il valore ottimo è verso lo 0;
    * **dominio**: MP6-CD $\in [0, 100]$;
    * **unità di misura**: percentuale;
    * **formula**: calcolata con un plugin di Maven.

B.2.3 Usabilità

- **MP7-NPSS Numero di Passi per il Setup del Server**
    * **descrizione**: misura il numero di passi che l'utente deve fare per installare il server in una macchina;
    * **obiettivo**: valutare la complessità della procedura d'installazione. Il valore ottimale è 1;
    * **dominio**: MP7-NPSS $\in \mathbb{N}\backslash\{0\}$;
    * **unità di misura**: numero di passi;
    * **formula**: N_E;
    * **input**:
        · N_E: numero di passi, intesi come comandi e operazioni, per configurare il prodotto da zero (numero naturale diverso da zero).

B.2.4 Efficienza

- **MP8-RST - Reazione agli Stress Test**
    * **descrizione**: misura la percentuale di stress test che il prodotto ha superato rispetto al totale degi stress test;
    * **obiettivo**: valutare la efficienza del prodotto. Questa metrica va massimizzata;
    * **dominio**: MP8-RST $\in [0, 100]$;
    * **unità di misura**: percentuale;

    * **formula**: $\frac{ST_S}{ST_T}$;
    * **input**:
        · ST_S: numero di stress test superati (numero naturale):
        · ST_T: numero totale di stress test (numero naturale diverso da zero).

B.2.5 Manutenibilità

- **MP9-CCM - Complessità Ciclomatica per Metodo**
    * **descrizione**: misura la complessità ciclomatica per metodo;
    * **obiettivo**: ridurre il tempo di testing e aumentare la manutenibilità del codice. Il valore ottimo è zero;
    * **target**: processi primari $\rightarrow$ sviluppo (sezione 2.2);
    * **dominio**: MW9-CMM $\in \mathbb{N}$;
    * **unità di misura**: unità;
    * **formula**: $E - N + 2P$;
    * **input**:
        · E: il numero di archi nel grafo di controllo del flusso (numero naturale);
        · N: il numero di nodi nel grafo di controllo del flusso (numero naturale);
        · P: il numero di componenti connesse nel grafo di controllo del flusso (numero naturale).
- **MP10-CS - Code Smell**
    * **descrizione**: misura il numero di debolezze del codice dal punto di vista della progettazione;
    * **obiettivo**: individuare quali parti necessitano di refactoring. Il valore ottimo è zero;
    * **dominio**: MP10-CS $\in \mathbb{Z}_+$;
    * **unità di misura**: unità;
    * **formula**: misurata con un plugin di Maven.
- **MP11-PAC - Profondità di Annidamento del Codice**

* **descrizione**: misura il livello di annidamento di ogni funzione nel codice;
* **obiettivo**: rendere più leggibile il codice;
* **dominio**: MP11-PAC $\in \mathbb{N}$;
* **unità di misura**: unità;
* **formula**: A_{med};
* **input**:
    - A_{med}: profondità media di annidamento del codice di ogni funzione.