NORME DI PROGETTO

THIREMA PROJECT

www.redroundrobin.site — redroundrobin.site@gmail.com

INFORMAZIONI SUL DOCUMENTO

Versione	2.3.0+b1.0
Uso	Interno
Stato	Approvato
Destinatari	Red Round Robin
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Redattori	Alessandro Tommasin
	Fouad Mouad
	Nicolò Frison
Verificatori	Fouad Mouad
	Mariano Sciacco
	Lorenzo Dei Negri
Approvazione	Giovanni Vidotto

Registro delle modifiche

Versione	Descrizione	Data	Autore	Ruolo
2.3.0+b0.14	Approvazione documento	2019-04-06	Giovanni Vidotto	Responsabile
2.2.1+b0.13	Aggiornamento e verifica sezioni §2.2.5, §3.1.5, §3.4.5, §3.5.5	2019-04-01	Nicolò Frison e Lorenzo Dei Negri	Amministratore
2.2.0+b0.12	Approvazione documento	2019-03-23	Giuseppe Vito Bitetti	Responsabile
2.1.1+b0.11	Aggiornamento e verifica sezioni §2.2.5, §3.1.5, §3.4.5, §3.5.5	2019-03-17	Alessandro Tommasin e Fouad Mouad	Amministratore
2.1.0+b0.11	Approvazione documento	2019-03-16	Lorenzo Dei Negri	Responsabile
2.0.2+b0.10	Aggiornamento e verifica sezioni §2.2.6, §3.1.6, §3.2.6, §3.5.6	2019-03-14	Nicolò Frison e Mariano Sciacco	Responsabile
2.0.1+b0.10	Aggiornamento e verifica sezione §2.2.4	2019-03-12	Fouad Mouad e Mariano Sciacco	Responsabile
2.0.0+b0.5	Approvazione documento	2019-02-05	Giuseppe Vito Bitetti	Responsabile
1.1.2+b0.4	Aggiornamento e verifica sezione §2, §3 e §4	2020-02-03	Alessandro Tommasin, Mariano Sciacco e Lorenzo Dei Negri	Amministratore e verificatore
1.1.1+b0.4	Ristrutturazione e verifica delle sezioni §2, §3 e §4	2020-01-30	Alessandro Tommasin, Giuseppe Vito Bitetti e Mariano Sciacco	Amministratore e verificatore
1.1.0+b0.4	Approvazione documento	2019-01-15	Alessandro Tommasin	Responsabile
1.0.2+b0.3	Aggiornamento sezione §3.5 e §3.7	2019-12-28	Nicolò Frison	Amministratore
1.0.1+b0.2	Verifica sezione §3.4, §3.5, §3.6, §3.7 e §3.9	2020-12-27	Alessandro Tommasin e Mariano Sciacco	Verificatore
1.0.0+b0.2	Approvazione documento	2020-12-24	Lorenzo Dei Negri	Responsabile

0.1.5+b0.1	Stesura e verifica sezione §3.9	2019-12-22	Giuseppe Vito Bitetti e Alessandro Tommasin	Amministratore e verificatore
0.1.4+b0.1	Stesura e verifica sezione §3.4, §3.5, §3.6 e §3.7	2019-12-18	Nicolò Frison e Giovanni Vidotto	Amministratore e verificatore
0.1.3+b0.1	Stesura e verifica sezione §4.2	2019-12-14	Giuseppe Vito Bitetti e Nicolò Frison	Amministratore e verificatore
0.1.2+b0.1	Stesura e verifica sezione §4.1	2019-12-10	Giovanni Vidotto e Mariano Sciacco	Amministratore e verificatore
0.1.1+b0.1	Stesura e verifica sezione §2	2019-12-08	Nicolò Frison e Alessandro Tommasin	Amministratore e verificatore
0.1.0+b0.1	Approvazione documento	2019-12-07	Mariano Sciacco	Responsabile
0.0.4+b0.0	Stesura e verifica sezione §3.8 e §3.2	2019-11-24	Fouad Mouad, Lorenzo Dei Negri e Mariano Sciacco	Amministratore e verificatore
0.0.3+b0.0	Stesura e verifica sezione §3.1	2019-11-22	Giovanni Vidotto e Lorenzo Dei Negri	Amministratore e verificatore
0.0.2+b0.0	Stesura e verifica sezione §1	2019-11-18	Giuseppe Vito Bitetti e Alessandro Tommasin	Amministratore e verificatore
0.0.1+b0.0	Creazione documento	2019-11-15	Nicolò Frison	Amministratore

Indice

1 Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di definire le regole su cui si basa il *way of working* del gruppo Red Round Robin per lo svolgimento del progetto. Le attività che possono essere trovate all'interno di questo documento sono state prese da processi appartenenti allo standard ISO/IEC 12207:1995. Tutti i membri del gruppo sono quindi tenuti a prendere visione di questo documento così da garantire uniformità e coesione all'interno del progetto.

1.2 Scopo del prodotto

Il capitolato C6 si pone l'obiettivo di creare una web-application che consenta l'analisi di grosse moli di dati ricevuti da sensori di natura eterogenea. Tale applicazione mette a disposizione un'interfaccia che permette di visualizzare alcuni dati di interesse od eventuali correlazioni tra i dati stessi. Infine, per ogni tipologia di dato è possibile assegnare il monitoraggio di ogni tipologia di dato ad un particolare ente.

1.3 Glossario e documenti esterni

Per evitare possibili ambiguità relative alle terminologie (che andranno indicate in MAIUSCOLETTO) utilizzate nei vari documenti, saranno adottati due simboli:

- una *D* a pedice per indicare il nome di un particolare documento.
- una *G* a pedice per indicare un termine che sarà presente nel GLOSSARIO v1.4.0+b1.0$_D$.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **capitolato d'appalto C6 - ThiReMa:**
  https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C6.pdf

1.4.2 Riferimenti informativi

- **Standard ISO/IEC 12207:1995:**
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- **documentazione git:** https://git-scm.com/docs
- **documentazione GitHub:** https://help.github.com/en/github
- **documentazione LaTeX:** https://www.latex-project.org/help/documentation/

- **Cost Variance:** http://acqnotes.com/acqnote/tasks/cost-variances
- **Schedule Variance:** http://acqnotes.com/acqnote/tasks/schedule-variances
- **Semantic Versioning:** https://semver.org/spec/v2.0.0.html

2 Processi primari

2.1 Fornitura

2.1.1 Scopo

Il processo di fornitura è indispensabile per:

- capire quali strumenti e competenze sono necessari per la realizzazione dei prodotti descritti nei capitolati proposti;
- redigere un documento che descriva nel dettaglio come il fornitore (il gruppo) intenda organizzare il lavoro che porterà alla realizzazione del prodotto oggetto del capitolato scelto;
- stabilire se il materiale prodotto dai membri del fornitore per soddisfare i compiti a loro assegnati faccia quanto atteso e sia di qualità.

2.1.2 Aspettative

Il gruppo si prefigge l'obiettivo di confrontarsi frequentemente col proponente, per capire se il lavoro svolto è in linea con quanto richiesto dal capitolato scelto, e chiarire eventuali requisiti ove necessario.

2.1.3 Descrizione

Il processo di fornitura si riassume in sei attività principali che vengono descritte sinteticamente di seguito.

2.1.4 Attività

2.1.4.1 Inizializzazione

Gli analisti del gruppo Red Round Robin devono effettuare una valutazione dei capitolati proposti tramite un documento che contenga uno studio di fattibilità. Il documento dovrà contenere, per ogni capitolato, una breve descrizione del capitolato stesso e delle tecnologie proposte, i suoi aspetti positivi e quelli negativi. Dovrà essere presente infine una conclusione che riassuma la motivazione per la quale si è scelto un capitolato rispetto ad un altro. Il contenuto del documento indicherà, alla fine, il capitolato scelto dal gruppo.

2.1.4.2 Preparazione della risposta

Il gruppo dovrà preparare una lettera di presentazione in cui si candiderà alla fornitura del prodotto implicato dal capitolato scelto.

2.1.4.3 Pianificazione

Il gruppo deve stipulare un piano per la gestione del progetto e della qualità; più nello specifico dovrà realizzare:

- un piano di progetto, contenente:
  - analisi e riscontro dei rischi;
  - modello di sviluppo scelto;
  - pianificazione coesa con il modello di sviluppo;
  - preventivo e consuntivo di periodo;
- un piano di qualifica, contenente:
  - metriche di qualità di processo;
  - metriche di qualità di prodotto;
  - specifiche dei test;
  - resoconto delle attività di verifica;
  - valutazioni di miglioramento.

2.1.4.4 Esecuzione e controllo

Il gruppo Red Round Robin dovrà seguire quanto descritto nei piani elencati in precedenza, monitorando i costi, le problematiche riscontrate, l'avanzamento nello sviluppo e la qualità.

2.1.4.5 Revisione e valutazione

Il gruppo Red Round Robin dovrà coordinare le revisioni delle attività svolte e le comunicazioni con il proponente ed il committente. Il gruppo dovrà inoltre eseguire la verifica e la validazione del prodotto garantendo che il prodotto sia conforme con quanto pattuito con il proponente ed il committente.

2.1.4.6 Rilascio e completamento

Il gruppo dovrà rilasciare il prodotto in maniera conforme a quanto pattuito con il proponente ed il committente.

2.1.5 Metriche

Il processo di fornitura non fa uso di metriche qualitative particolari.

2.1.6 Strumenti

Per il processo di fornitura non sono stati individuati particolari strumenti da impiegare.

2.2 Sviluppo

2.2.1 Scopo

Il processo di sviluppo definisce i compiti e le attività da intraprendere per ottenere il prodotto finale richiesto dal proponente.

2.2.2 Aspettative

Per una corretta implementazione di questo processo è necessario fissare:

- obiettivi di sviluppo;
- vincoli tecnologici e di design.

Il prodotto finale deve rispettare i requisiti e le aspettative del proponente, superando i test definiti dalle norme di qualità.

2.2.3 Descrizione

Il processo di sviluppo, secondo lo standard ISO/IEC 12207:1995, si articola nelle seguenti attività:

- analisi dei requisiti;
- progettazione;
- codifica.

2.2.4 Attività

Di seguito verranno analizzate dettagliatamente le attività menzionate nella sezione precedente.

2.2.4.1 Analisi dei requisiti

Scopo

Gli analisti si occupano di stilare il documento di analisi dei requisiti, il cui scopo è definire ed elencare tutti i requisiti del capitolato. Il documento finale conterrà:

- descrizione generale del prodotto;
- argomentazioni precise ed affidabili per i progettisti;
- casi d'uso rappresentati tramite diagrammi UML;
- funzionalità e requisiti concordi con le richieste del cliente;
- tracciamento dei requisiti individuati.

Aspettative

Creazione del documento formale contente tutti i requisiti richiesti dal proponente per la realizzazione del capitolato.

Classificazione dei Requisiti

Al fine di facilitarne la consultazione e comprensione, i requisiti saranno classificati ed identificati univocamente secondo il seguente schema identificativo:

R[Priorità]-[Tipologia]-[Identificativo]

Dove:

- **R:** indica che si tratta di un requisito;
- **Priorità:** assume, a seconda del grado di priorità, i valori:
  - **A:** obbligatorio, cioè strettamente necessario;
  - **B:** desiderabile, cioè non strettamente necessario;
  - **C:** opzionale, cioè contrattabile in corso d'opera;
- **Tipologia:** a seconda del tipo di requisito, assume i valori:
  - **F:** funzionale;
  - **P:** prestazionale;
  - **Q:** qualitativo;
  - **V:** vincolo;
- **Identificativo:** numero progressivo per contraddistinguere il requisito, in forma gerarchica padre-figlio strutturato come segue:

[codicePadre].[codiceFiglio]

Classificazione dei casi d'uso

Gli analisti, dopo la stesura dei requisiti, hanno anche il compito di identificare ed elencare i casi d'uso. Ognuno di essi è identificato, in maniera univoca, secondo il seguente schema identificativo:

UC[codiceCaso].[codiceSottoCaso].[codiceSottoSottoCaso]

Ogni caso d'uso oltre al codice di identificazione deve contenere, integralmente o parzialmente, i seguenti campi:

- **diagrammi UML:** diagrammi realizzati usando la versione 2.0 del linguaggio;
- **attori primari:** attori principali del caso d'uso;
- **attori secondari:** attori secondari del caso d'uso;

- **descrizione:** breve descrizione del caso d'uso;
- **attori secondari:** attori secondari del caso d'uso;
- **estensioni:** eventuali estensioni coinvolte;
- **inclusioni:** eventuali inclusioni coinvolte;
- **precondizione:** condizioni che devono essere soddisfatte perché si verifichino gli eventi del caso d'uso;
- **post-condizione:** condizioni che devono essere soddisfatte dopo il verificarsi degli eventi del caso d'uso;
- **scenario principale:** flusso degli eventi, in forma di elenco numerato, con eventuale riferimento ad ulteriori casi d'uso.

2.2.4.2 Progettazione

Scopo

L'attività di progettazione precede quella di codifica ed avviene successivamente all'analisi dei requisiti.
In questa attività i progettisti hanno il compito di definire una soluzione del problema che sia soddisfacente per gli STAKEHOLDER$_G$.
Lo scopo della progettazione è definire l'architettura logica del prodotto da sviluppare e deve permettere di:

- garantire l'efficacia del prodotto, soddisfacendo tutti i requisiti individuati nell'attività di analisi, attraverso un sistema di qualità che persegua il principio di correttezza per costruzione;
- garantire l'efficienza nella realizzazione del prodotto, impiegando parti riusabili con specifiche chiare, organizzate in modo da facilitarne la manutenzione, e realizzabili ottimizzando l'utilizzo delle risorse disponibili;
- gestire la complessità del sistema, suddividendolo fino ad ottenere delle parti di complessità trattabile, che possano essere fornite in ingresso all'attività di codifica come singoli compiti individuali che siano, quindi, fattibili, rapidi e verificabili.

Aspettative

Le aspettative della progettazione è la definizione l'architettura logica del prodotto, la quale dovrà godere delle seguenti qualità:

- **sufficienza:** capace di soddisfare tutti i requisiti;
- **comprensibilità:** capibile da tutti gli stakeholder;
- **modularità:** suddivisibile in parti chiare e ben distinte, riducendo la dipendenza tra le parti stesse, per ridurre i cambiamenti esterni causati da modifiche interne ad una singola parte;
- **robustezza:** capace di sopportare ingressi diversi da parte dell'utente e dell'ambiente;

- **flessibilità:** modificabile a costo contenuto al variare dei requisiti;
- **efficienza nella gestione delle risorse;**
- **affidabilità:** se utilizzata correttamente, garantisce di ottenere il risultato desiderato;
- **disponibilità:** richiede un tempo di indisponibilità limitato o nullo per effettuare la manutenzione;
- **sicurezza:** non presenta malfunzionamenti gravi nè vulnerabilità alle intrusioni;
- **semplicità:** composta di parti contenenti solamente il necessario e nulla di superfluo;
- **incapsulazione:** composta di parti la cui struttura interna non è visibile da fuori, in modo da ridurre le dipendenze indotte sull'esterno e quindi facilitare la manutenzione;
- **coesione:** composta da parti raggruppate per obiettivi comuni, in modo da ridurre l'interdipendenza tra le componenti e quindi aumentare la manutenibilità e la comprensibilità;
- **basso accoppiamento:** composta da parti distinte con dipendenza bassa o nulla le une dalle altre.

Descrizione

Per perseguire la qualità nella progettazione dell'architettura è necessario seguire le seguenti regole:

- ogni attività svolta deve essere coerente con quanto individuato nell'attività di analisi dei requisiti;
- se possibile, prediligere sempre l'utilizzo di opportuni DESIGN PATTERN$_G$;
- evitare l'utilizzo dell'ereditarietà tra classi concrete, prediligendo sempre l'uso di classi astratte e/o interfacce;
- perseguire sempre il principio dell'incapsulamento e dell'*information hiding*, limitando il più possibile la visibilità dei dettagli implementativi;
- assegnare sempre nomi significativi e parlanti a package, classi, metodi e variabili;
- evitare la definizione di dipendenze circolari tra classi;
- se possibile, definire sempre relazioni tra componenti con il minor grado di dipendenza realizzabile.

L'attività di progettazione si articola nelle seguenti due parti:

- **progettazione architetturale:** definizione delle specifiche, ad alto livello, dell'architettura e delle componenti del prodotto, delle loro interazioni con le restanti parti del sistema e dei test di integrazione; al termine di questo periodo di progettazione si ottiene la technology baseline del progetto;
- **progettazione di dettaglio:** definizione delle specifiche di dettaglio dell'architettura e delle componenti del prodotto, fino ad arrivare a singole unità, a partire dalla technology baseline; inoltre sono definiti l'insieme dei diagrammi UML che descrivono l'architettura e le sue componenti, scomposte in unità, e i test necessari per la loro verifica; al termine di questo periodo di progettazione si ottiene la product baseline del progetto.

Technology baseline

La technology baseline deve includere:

- **PROOF OF CONCEPT$_G$:** primo eseguibile del sistema con funzione dimostrativa, deve fornire una base di integrazione delle tecnologie necessarie al prodotto finale;
- **tecnologie utilizzate:** descrizione dettagliata delle tecnologie impiegate nello sviluppo del progetto, con particolare enfasi sui pregi e i difetti riscontrati;
- **test di integrazione:** definizione dei test eseguiti per verificare che le varie componenti del sistema, una volta integrate insieme, interagiscano in modo corretto e in conformità con quanto richiesto dai requisiti;
- **tracciamento delle componenti:** associazione tra requisiti e componenti che li soddisfano.

Product baseline

La product baseline deve includere:

- **DESIGN PATTERN$_G$:** descrizione dei design pattern utilizzati nella definizione dell'architettura, per la soluzione progettuale a problemi *ricorrenti* riscontrati; ogni design pattern deve essere opportunamente descritto, con una spiegazione del suo significato, ed accompagnato da un diagramma che ne mostri la struttura;
- **DIAGRAMMI UML$_G$:** diagrammi realizzati in linguaggio UML versione 2.0, utilizzati per rendere più chiare le soluzioni progettuali adottate; essi si suddividono in:
  - **diagrammi dei package:** descrivono le dipendenze tra classi raggruppate in diversi package, ossia raggruppamenti di elementi in un'unità di livello più alto;
  - **diagrammi delle classi:** rappresentano le classi che compongono il sistema, assieme ai loro attributi e metodi, e le relazioni di dipendenza che sussistono tra loro;
  - **diagrammi di sequenza:** descrivono la collaborazione di un gruppo di oggetti che devono realizzare un determinato comportamento, attraverso una sequenza di azioni e di scelte definite;
  - **diagrammi delle attività:** descrivono la logica procedurale di un processo o flusso di operazioni, aiutando a descrivere gli aspetti dinamici dei casi d'uso.
- **test di unità:** definizione dei test eseguiti per verificare che il funzionamento delle varie classi e metodi che implementano il sistema software sia corretto e conforme ai requisiti;
- **tracciamento delle classi:** associazione tra requisiti e classi che li soddisfano.

Segue un'analisi più approfondita dei diagrammi UML e delle caratteristiche che devono avere.

Diagrammi dei package

I diagrammi dei package vengono utilizzati per raggruppare gli elementi interni ad un elemento di più alto livello. Ogni package identifica un diverso spazio dei nomi, il quale permette di dare un nome qualificato a tutti gli elementi in esso contenuti.

Questi diagrammi permettono di rappresentare le dipendenze tra i diversi package che li compongono; ciò consente di individuare le dipendenze circolari, in modo da rimuoverle, e di capire quali elementi del sistema dovranno essere i più stabili, verificando quali hanno più dipendenze entranti. Inoltre, tramite i diagrammi dei package è possibile controllare la complessità strutturale del sistema.
Rispetto ad altri package esterni, gli elementi interni di un determinato package possono avere visibilità pubblica oppure privata.

Diagrammi delle classi

I diagrammi delle classi descrivono i tipi di oggetto presenti nel sistema e le relazioni presenti tra essi.
Le classi rappresentate in questi diagrammi presentano le seguenti caratteristiche:

- **nome:** nome della classe;
- **attributi (opzionali):** rappresentano lo stato interno della classe; sono identificati dalle seguenti proprietà:
  - **visibilità:** visibilità dell'attributo rispetto ad altre classi, può essere pubblica, protetta o privata;
  - **nome:** nome dell'attributo;
  - **tipo:** tipo di dato dell'attributo;
  - **molteplicità (opzionale):** occorrenze dell'attributo nella classe;
  - **default (opzionale):** valore predefinito dell'attributo;
  - **proprietà aggiuntive (opzionali);**
- **operazioni (opzionali):** rappresentano le azioni eseguibili dalla classe; sono identificate dalle seguenti proprietà:
  - **visibilità:** visibilità dell'operazione rispetto ad altre classi, può essere pubblica, protetta o privata;
  - **nome:** nome dell'operazione;
  - **parametri:** lista dei parametri dell'operazione; ogni parametro è identificato delle seguenti proprietà:
    - **direzione (opzionale):** modalità di accesso al parametro, può essere in lettura, in scrittura o in entrambe;
    - **nome:** nome del parametro;
    - **tipo:** tipo del parametro;
    - **default (opzionale):** valore predefinito del parametro;
  - **ritorno:** tipo di ritorno dell'operazione;
  - **proprietà aggiuntive (opzionali);**
- **relazioni (opzionali):** rappresentano il grado di dipendenza tra due classi:
  - **dipendenza:** la classe utilizza un oggetto della classe con cui si relaziona;
  - **associazione:** la classe crea ed utilizza un oggetto della classe con cui si relaziona;
  - **aggregazione:** la classe possiede come attributo un riferimento ad un oggetto condiviso della classe con cui si relazione;

  - **composizione:** la classe possiede come attributo un oggetto della classe con cui si relaziona;
  - **generalizzazione:** la classe è un istanza che specializza la classe con cui si relazione, ereditandone attributi ed operazioni.

Oltre alle classi, in questi diagrammi possono essere utilizzati i seguenti costrutti:

- **classe astratte:** rappresenta una classe di cui non è possibile instanziare alcun oggetto, in quanto possiede almeno un'operazione astratta (operazione dichiarata ma non definita); le classi che specializzano una classe astratta, per non essere a loro volta astratte, devono obbligatoriamente prevedere una definizione delle operazioni astratte ereditate;
- **interfaccia:** rappresenta un costrutto simile ad una classe astratta, ma a differenza di quest'ultima non può possedere attributi e tutte le sue operazioni sono obbligatoriamente astratte.

Diagrammi di sequenza

I diagrammi di sequenza descrivono la realizzazione di un determinato comportamento tramite la collaborazione di più oggetti che si scambiano messaggi, mostrandone il funzionamento della loro interazione nel tempo.
I costrutti utilizzati in questi diagrammi sono i seguenti:

- **partecipante:** rappresenta un oggetto che detiene il flusso di esecuzione e collabora alla realizzazione di un comportamento; un partecipante è composto di due parti:
  - **nome:** nome dell'oggetto partecipante;
  - **barra di attivazione:** indicazione della durata del periodo di tempo durante il quale il partecipante è attivo;
- **messaggio:** rappresenta un'operazione di un partecipante che viene chiamata da parte di un altro partecipante e i dati scambiati tra i due; un messaggio può essere di una delle seguenti tipologie:
  - **sincrono:** messaggio di chiamata in cui il partecipante chiamante attende la risposta del partecipante chiamato prima di proseguire la sua esecuzione;
  - **asincrono:** messaggio di chiamata in cui il partecipante chiamante non attende la risposta del partecipante chiamato, ma prosegue la sua esecuzione subito dopo la chiamata;
  - **ritorno:** messaggio di ritorno riferito ad un precedente messaggio di chiamata;
  - **creazione:** messaggio di creazione di un nuovo partecipante da parte del partecipante chiamante;
  - **distruzione:** messaggio di distruzione di un partecipante da parte del partecipante chiamante;
- **frame di interazione:** rappresenta un ciclo o una condizione che coinvolge più messaggi e parti delle barre di attivazione di più partecipanti; un frame è caratterizzato dalle seguenti due proprietà:
  - **guardia:** indica la condizione di attivazione del frame, posta in corrispondenza del partecipante coinvolto;

  - **etichetta:** indica la tipologia del frame; esistono sette tipi di frame di interazione e sono etichettati nel seguente modo:
    * **alt:** alternativa (tra più frame), è eseguito solo il frame per cui la guardia è verificata;
    * **opt:** opzionale, il frame è eseguito solo se la guardia è verificata;
    * **par:** parallelo, ogni frame è eseguito in parallelo;
    * **loop:** ciclo, il frame può essere eseguito più volte, in base al verificarsi della guardia;
    * **region:** regione critica, il frame può essere eseguito da un solo flusso di esecuzione alla volta;
    * **neg:** negativo, il frame rappresenta un'interazione non valida;
    * **ref:** riferimento, il frame si riferisce ad un'interazione definita in un altro diagramma;
    * **sd:** diagramma di sequenza, il frame comprende un intero diagramma di sequenza.

Tramite i diagrammi di sequenza è possibile definire due tipi di controllo dell'elaborazione durante la collaborazione tra i diversi partecipanti:

- **centralizzato:** un unico partecipante concentra su di sè la responsabilità della chiamata dei messaggi verso tutti gli altri partecipanti, governando l'intera elaborazione; diventa quindi un punto critico dell'architettura del sistema;
- **distribuito:** ogni partecipante ha dei compiti ben definiti e la responsabilità della chiamata dei messaggi verso gli altri partecipanti coinvolti.

Diagrammi delle attività

I diagrammi delle attività descrivono la logica dei processi che compongono l'applicazione software sviluppata. Permettono di rappresentare la parte dinamica dei casi d'uso, mostrando le attività che si possono presentare e le sequenze in cui le loro azioni vengono svolte.
I costrutti utilizzati in questi diagrammi sono i seguenti:

- **nodo iniziale:** rappresenta il punto d'inizio dell'esecuzione dell'attività;
- **nodo di fine flusso:** rappresenta un punto di terminazione di un percorso di esecuzione; non causa la terminazione dell'esecuzione dell'attività, in quanto essa può continuare su altri percorsi;
- **nodo finale:** rappresenta il punto di terminazione dell'esecuzione dell'attività;
- **activity:** rappresenta un'azione all'interno dell'attività, identificata da una breve descrizione;
- **subactivity:** rappresenta una sotto-attività utilizzata per descrivere un'azione che ne comprende altre al suo interno; ogni sotto-attività può essere utilizzata come una singola azione all'interno del diagramma, in modo da non aumentarne troppo la dimensione e la complessità, e non comprometterne, quindi, la gestione e la comprensione; tuttavia deve essere sempre realizzato un diagramma della sotto-attività che ne illustri l'input, l'output e le azioni contenute;
- **pin:** rappresenta un parametro prodotto o consumato da un'azione, con l'indicazione del suo tipo;

- **fork:** rappresenta un punto d'inizio di un elaborazione parallela all'interno dell'attività;
- **join:** rappresenta un punto di sincronizzazione tra più flussi di esecuzione parallela generati in seguito ad un fork;
- **branch:** rappresenta il punto di scelta di uno tra i possibili percorsi di esecuzione disponibili, in base alla condizione di guardia associata a ciascuno di essi;
- **merge:** rappresenta un punto di unione dei diversi percorsi di esecuzione (non paralleli) generati in seguito ad un branch;
- **segnale:** rappresenta un evento esterno, generato in modo non bloccante e catturato in modo bloccante, all'interno dell'attività;
- **timeout:** rappresenta un attesa bloccante all'interno dell'attività, di cui deve essere specificata la durata e l'unità di misura;
- **evento ripetuto:** rappresenta un evento che viene generato ad intervalli regolari all'interno dell'attività, di cui deve essere specificata la durata e l'unità di misura;
- **swimlane:** rappresenta una partizione che fornisce una singola responsabilità all'esecuzione delle azioni all'interno di un'attività;
- **regione di espansione:** rappresenta una sequenza di azioni che vengono ripetute sugli elementi di una collezione, presenti in una lista degli argomenti della regione di espansione.

2.2.4.3 Codifica

Scopo

Lo scopo di questa attività è di normare la concretizzazione del prodotto attraverso la programmazione. Gli sviluppatori, durante la fase di implementazione, dovranno attenersi alle norme sotto elencate.

Aspettative

L'obiettivo è lo sviluppo del software richiesto dal proponente, utilizzando le norme di programmazione stabilite, in modo da:

- ottenere codice leggibile ed uniforme per i programmatori;
- agevolare le fasi di manutenzione, verifica e validazione;
- fornire un prodotto conforme alle richieste prefissate dal proponente;
- realizzare un prodotto di qualità.

Descrizione

Il codice scritto dovrà rispettare le norme stabilite in questo documento, con il fine di perseguire una buona livello di qualità, conforme alle metriche stabilite.

Stile di codifica

Per garantire l'uniformità del codice, in base al linguaggio utilizzato, ciascuno sviluppatore dovrà attenersi a quanto segue:

- **Java:** si è deciso di seguire le linee guida fornite da GOOGLE$_G$ per il linguaggio, in particolare:
  - **parentesizzazione:** le parentesi graffe verranno utilizzate sempre, anche se il corpo del blocco è vuoto o con una sola istruzione; inoltre, si segue lo stile Kernighan and Ritchie per quanto riguarda l'utilizzo degli spazi e dei capo-riga;

```
return () -> {
  while (condition()) {
    method();
  }
};

  return new MyClass() {
  @Override public void method() {
    if (condition()) {
      try {
        something();
      } catch (ProblemException e) {
        recover();
      }
    } else if (otherCondition()) {
      somethingElse();
    } else {
      lastThing();
    }
  }
};
```

Snippet 1: Esempio per la sintassi Java

  - **indentazione:** i blocchi del codice devono essere sempre indentati, per ciascun livello, con tabulazione la cui larghezza sia impostata a due (2) spazi; ogni programmatore dovrà configurare il proprio editor di testo secondo questa regola (l'indentazione dei commenti non viene considerata);
  - **verbosità:** ogni istruzione non potrà essere più lunga di cento (100) caratteri;
  - **univocità dei nomi:** metodi e variabili devono avere nomi univoci che ne descrivano il più possibile la funzione; la prima lettera deve essere sempre minuscola e, nel caso in cui il/la metodo/variabile sia una concatenazione di più parole, i programmatori devo attenersi al LOWERCAMELCASE$_G$. per quanto riguarda le classi si applicano le regole esposte in precedenza, tranne per la lettera iniziale del nome, che sarà maiuscola, in linea con UPPERCAMELCASE$_G$;
  - **ricorsione:** l'uso della ricorsione è da evitare ove possibile, in quanto aumenterebbe la complessità computazionale del codice;
  - **lingua:** la lingua con cui viene scritto il codice deve essere l'inglese, mentre per i commenti può essere usato l'italiano;

Ulteriori dettagli sono disponibili alla risorsa online Google Java Style Guide;

- **JavaScript:** si è deciso di seguire le linee guida fornite da GOOGLE$_G$ per il linguaggio, in particolare:
  - **parentesizzazione:** le parentesi graffe verranno utilizzate sempre, anche se il corpo del blocco è vuoto o con una sola istruzione; inoltre, si segue lo stile Kernighan and Ritchie per quanto riguarda l'utilizzo degli spazi e dei capo-riga;

```
class InnerClass {
  constructor() {}

  /** @param {number} foo */
  method(foo) {
    if (condition(foo)) {
      try {
        // Note: this might fail.
        something();
      } catch (err) {
        recover();
      }
    }
  }
}
```

Snippet 2: Esempio per la sintassi JavaScript

  - **indentazione:** i blocchi del codice devono essere indentati, per ciascun livello, con tabulazione la cui larghezza sia impostata a due (2) spazi; ogni programmatore dovrà configurare il proprio editor di testo secondo questa regola (l'indentazione dei commenti non viene considerata);
  - **verbosità:** ogni istruzione non potrà essere più lunga di ottanta (80) caratteri;
  - **univocità dei nomi:** metodi e variabili devono avere nomi univoci che ne descrivano il più possibile la funzione; la prima lettera deve essere sempre minuscola e, nel caso in cui il/la metodo/variabile sia una concatenazione di più parole, i programmatori devo attenersi al LOWERCAMELCASE$_G$; per quanto riguarda le classi si applicano le regole esposte in precedenza, tranne per la lettera iniziale del nome, che sarà maiuscola, in linea con UPPERCAMELCASE$_G$;
  - **ricorsione:** l'uso della ricorsione è da evitare ove possibile, in quanto aumenterebbe la complessità computazionale del codice;
  - **lingua:** la lingua con cui viene scritto il codice deve essere l'inglese, mentre per i commenti può essere usato l'italiano;

Ulteriori dettagli sono disponibili alla risorsa online Google JavaScript Style Guide;

- **PHP:** si è deciso di utilizzare le linee guida fornite dallo standard PSR-12 per il linguaggio, in particolare:
  - **parentesizzazione:** le parentesi graffe verranno utilizzate sempre, anche se il corpo del blocco è vuoto o con una sola istruzione; inoltre, si segue lo stile Kernighan and Ritchie per quanto riguarda l'utilizzo degli spazi e dei capo-riga;

```php
<?php

    declare(strict_types=1);

    namespace Vendor\Package;

    use Vendor\Package\{ClassA as A, ClassB, ClassC as C};
    use Vendor\Package\SomeNamespace\ClassD as D;

    use function Vendor\Package\{functionA, functionB, functionC
   };

    use const Vendor\Package\{ConstantA, ConstantB, ConstantC};

    class Foo extends Bar implements FooInterface
    {
        public function sampleFunction(int $a, int $b = null):
   array
        {
            if ($a === $b) {
                bar();
            } elseif ($a > $b) {
                $foo->bar($arg1);
            } else {
                BazClass::bar($arg2, $arg3);
            }
        }

        final public static function bar()
        {
            // method body
        }
    }
```

Snippet 3: Esempio per la sintassi PHP

- **indentazione:** i blocchi del codice devono essere indentati, per ciascun livello, con tabulazione la cui larghezza sia impostata a quattro (4) spazi; ogni programmatore dovrà configurare il proprio editor di testo secondo questa regola (l'indentazione dei commenti non viene considerata);
- **verbosità:** ogni istruzione non potrà essere più lunga di centoventi (120) caratteri;
- **univocità dei nomi:** metodi e variabili devono avere nomi univoci e che ne descrivano il più possibile la funzione; la prima lettera deve essere sempre minuscola e, nel caso in cui il/la metodo/variabile sia una concatenazione di più parole, i programmatori devo attenersi al LowerCamelCase$_G$; per quanto riguarda le classi si applicano le regole esposte in precedenza tranne per la lettera iniziale del nome, che sarà maiuscola in linea con UpperCamelCase$_G$;
- **ricorsione:** l'uso della ricorsione è da evitare ove possibile, in quanto aumenterebbe la complessità computazionale del codice;

  - **lingua:** la lingua con cui viene scritto il codice deve essere l'inglese, mentre per i commenti può essere usato l'italiano;

Ulteriori dettagli sono disponibili alla risorsa online PSR PHP Style Guide.

Linee guida

Per quanto riguarda la verifica dei linguaggi sopra detti, rispetto alle linee guida di riferimento, ci si riferisce a:

- **Checkstyle:** usato per la verifica della sintassi del linguaggio Java, con le regole riportate dal Google Java Style Guide;
- **Prettier, ESLint:** usato per la verifica della sintassi del linguaggio JavaScript, con le regole riportate dal Google JavaScript Style Guide;
- **PHP CodeSniffer:** usato per la verifica del linguaggio PHP, con le regole riportate dal PSR PHP Style Guide;
- **W3C:** usato per HTML5 e CSS3, in quanto il gruppo si attiene alle regole imposte dal World Wide Web Consortium; per assicurarsi il rispetto di tali norme il codice prodotto verrà verificato tramite i validatori offerti dal W3C:
  - Markup Validation Service;
  - CSS Validation Service.

2.2.5 Metriche

2.2.5.1 QC-2 Sicurezza

Scopo

La sicurezza viene analizzata per fornire al cliente un prodotto che sia privo di vulnerabilità note, sia a livello di librerie, che a livello di contenuti.

Introduzione alle Metriche

Per la sicurezza si è deciso di utilizzare le seguenti metriche:

- QM-PROD-4 Numero di vulnerabilità rilevate (NVUL);
- QM-PROD-5 Tempo di risoluzione vulnerabilità (TVUL).

QM-PROD-4 Numero di vulnerabilità rilevate (NVUL)

- **Descrizione:** la metrica permette di rilevare il numero di vulnerabilità di sicurezza, attraverso SonarCloud$_G$, per un componente software;
- **Unità di misura:** la metrica è espressa con un numero intero;

- **Fonte:** il dato viene recuperato attraverso le statistiche di SonarCloud$_G$;
- **Risultato:** il risultato delle rilevazioni ha i seguenti significati:
  - se il risultato è pari a 0, allora non sono state rilevate vulnerabilità nell'applicazione e/o nelle sue dipendenze;
  - se il risultato è maggiore di 0, allora sono state rilevate vulnerabilità nell'applicazione e/o nelle sue dipendenze.

QM-PROD-5 Tempo di risoluzione vulnerabilità (TVUL)

- **Descrizione:** La metrica permette di comprendere la stima temporale per correggere le vulnerabilità di sicurezza rilevate attraverso SonarCloud$_G$, per un componente software;
- **Unità di misura:** La metrica è espressa in minuti;
- **Fonte:** Il dato viene recuperato attraverso le statistiche di SonarCloud$_G$;
- **Risultato:** Il risultato delle rilevazioni ha i seguenti significati:
  - se il risultato è pari a 0 allora non sono state scoperte vulnerabilità nell'applicazione e/o nelle sue dipendenze;
  - se il risultato è maggiore di 0 allora sono state rilevate delle vulnerabilità che hanno un tempo di risoluzione stimato pari a quello riportato.

2.2.5.2 QC-3 Affidabilità

Scopo

Durante lo sviluppo si vuole monitorare l'affidabilità e la correttezza, così come la sua tolleranza agli errori:

- **maturità:** il software deve essere in grado di evitare il verificarsi di errori e/o malfunzionamenti derivanti dalla sua esecuzione;
- **tolleranza degli errori:** il prodotto è in grado di mantenere un livello minimo di prestazioni predeterminate anche in presenza di malfunzionamenti e/o usi impropri di esso;
- **recuperabilità:** il software, in seguito ad un errore e/o malfunzionamento, deve essere in grado di ripristinare uno stato di usabilità in un arco di tempo definito e di recuperare eventuali dati persi durante il suddetto lasso di tempo;
- **aderenza:** descrive la capacità del prodotto di aderire alle specifiche relative all'affidabilità.

Introduzione alle Metriche

Per l'affidabilità si è deciso di utilizzare le seguenti metriche:

- QM-PROD-6 Numero di bug rilevati (BUGR);
- QM-PROD-7 Tempo stimato risoluzione bug (TBUG).

QM-PROD-6 Numero di bug rilevati (BUGR)

- **Descrizione:** la metrica permette di rilevare il numero di bug che sono stati scoperti attraverso SonarCloud$_G$, per un componente software;
- **Unità di misura:** la metrica è espressa in numero intero positivo, a partire da 0;
- **Fonte:** il dato viene recuperato attraverso le statistiche di SonarCloud$_G$;
- **Risultato:** il risultato delle rilevazioni ha i seguenti significati:
  - se il risultato è pari a 0 allora non sono stati rilevati bug;
  - se il risultato è maggiore di 0 allora sono stati rilevati dei bug.

QM-PROD-7 Tempo stimato risoluzione bug (TBUG)

- **Descrizione:** la metrica permette di rilevare il tempo stimato per la risoluzione dei bug che sono stati scoperti attraverso SonarCloud$_G$, per un componente software;
- **Unità di misura:** la metrica è espressa in minuti;
- **Fonte:** il dato viene recuperato attraverso le statistiche di SonarCloud$_G$;
- **Risultato:** il risultato delle rilevazioni ha i seguenti significati:
  - se il risultato è pari a 0 allora non sono stati rilevati bug;
  - se il risultato è maggiore di 0 allora sono stati rilevati dei bug che hanno un tempo di risoluzione stimato pari a quello riportato.

2.2.5.3 QC-4 Efficienza

Scopo

Si vuole misurare l'efficienza del prodotto in modo da fornire all'utente un software usabile, piacevole e veloce:

- **comportamento nel tempo:** garanzia di tempi di elaborazione accettabili da parte del prodotto;
- **utilizzo di risorse:** utilizzo non eccessivo delle risorse a disposizione.

Introduzione alle Metriche

Per l'efficienza si è deciso di utilizzare la seguente metrica:

- QM-PROD-8 Risposta media (RM).

QM-PROD-8 Risposta media (RM)

- **Descrizione:** la metrica RM punta a misurare l'efficienza di elaborazione del prodotto in modo da fornire all'utente un'esperienza piacevole di utilizzo;
- **Unità di misura:** la metrica è espressa millisecondi (*ms*);
- **Formula:** la formula della metrica è la seguente:

$$\text{RM} = \frac{\sum_{n=1}^{z} \text{tempo di risposta in ms}}{z}$$

  dove z è il numero di misurazioni effettuate;
- **Risultato:** il risultato della formula indica il tempo medio di risposta del prodotto:
  - se è pari a 0, allora non è stata fatta alcuna misurazione poiché non è stato possibile;
  - se è maggiore di 0, allora è stato possibile eseguire una misurazione.

2.2.5.4 QC-5 Usabilità

Scopo

Si vuole misurare l'usabilità del prodotto in modo da fornire all'utente un'esperienza piacevole durante il suo utilizzo, nonché fornire un prodotto facile da apprendere ed utilizzare:

- **comprensibilità:** determina la facilità di utilizzo e di comprensione del prodotto e delle sue funzionalità da parte dell'utente;
- **apprendibilità:** definisce il livello di impegno richiesto, da parte dell'utilizzatore, per imparare ad usare il prodotto;
- **operabilità:** stabilisce il grado con cui il software riesce a mettere il suo utilizzatore in condizione di sfruttare il prodotto per i suoi fini;
- **attrattiva:** la proprietà del software di produrre un'esperienza d'uso gradevole per l'utente.

Introduzione alle Metriche

Per l'usabilità si è deciso di utilizzare le seguenti metriche:

- QM-PROD-9 Profondità dell'albero delle azioni (PAA);
- QM-PROD-10 Profondità dell'albero delle pagine (PAP).

QM-PROD-9 Profondità dell'albero delle azioni (PAA)

- **Descrizione:** la metrica PAA permette di misurare l'operabilità e la comprensibilità del prodotto da parte dell'utente; essa misura il numero massimo di azioni effettuate dall'utente prima di poter arrivare al suo obiettivo; viene misurato per il sito web in base alle tipologie di utente previste;

- **Unità di misura:** la metrica è espressa con un numero intero;
- **Formula:** la formula della metrica è la seguente: *PAA = numero di click* dove ogni click corrisponde ad un'azione;
- **Risultato:** il risultato della formula ha i seguenti significati:
  - se il risultato è minore della soglia prevista, allora l'obiettivo potrà essere portato a termine con maggiore semplicità;
  - se il risultato è maggiore della soglia prevista, allora l'obiettivo potrà essere portato a termine con maggiore difficoltà;

QM-PROD-10 Profondità dell'albero delle pagine (PAP)

- **Descrizione:** la metrica PAP permette di misurare l'apprendibilità, l'operabilità e l'attrattiva del prodotto dal punto di vista del cliente; essa misura il numero massimo di pagine visitate dall'utente prima di poter arrivare al suo obiettivo; viene misurato per il sito web in base alle tipologie di utente previste;
- **Unità di misura:** la metrica è espressa tramite un numero intero;
- **Formula:** la formula della metrica è la seguente: *PAP = numero di pagine*
- **Risultato:** il risultato della formula ha i seguenti significati:
  - un numero minore di pagine, conduce a minore smarrimento dell'utente nel sito;
  - un numero troppo di alto di pagine, può condurre un maggiore senso di smarrimento dell'utente nel sito.

2.2.5.5 QC-6 Manutenibilità

Scopo

La manutenibilità viene monitorata per essere in grado di fornire un prodotto modificabile e migliorabile, in modo da poter estendere facilmente sia la vita del prodotto stesso, sia le sue funzionalità.

- **analizzabilità:** determina la facilità con cui è possibile analizzare e localizzare un errore all'interno del codice;
- **modificabilità:** definisce la capacità del prodotto di apportare una modifica o una estensione;
- **stabilità:** il software deve essere un grado di essere usato anche in caso le modifiche apportate siano errate;
- **testabilità:** determina la capacità del software di essere testato facilmente per fornire una validazione delle modifica apportate.

Introduzione alle Metriche

Per la manutenibilità si è deciso di utilizzare le seguenti metriche:

- QM-PROD-11 Complessità ciclomatica media (COCIM);
- QM-PROD-12 Complessità della classe (CCLA);
- QM-PROD-13 Numero di code smell rilevati (NCS);
- QM-PROD-14 Tempo di risoluzione code smell (TCS);
- QM-PROD-15 Percentuale di duplicazione del Codice (DUPC);
- QM-PROD-16 Numero di violazioni standard di codifica (NVSC).

QM-PROD-11 Complessità ciclomatica media (COCIM)

- **Descrizione:** la metrica viene usata per rilevare la complessità ciclomatica media di tutte le classi e di tutti i metodi realizzati nel programma; da questa si comprende la quantità delle condizioni realizzate e la loro relativa complessità media;
- **Unità di misura:** la metrica è espressa con un numero decimale;
- **Fonte:** il dato viene recuperato attraverso le statistiche di SONARCLOUD$_G$;
- **Risultato:** il risultato rilevato ha i seguenti significati:
  - se il risultato è pari a 0, allora non sono presenti dei punti condizionali del programma;
  - se il risultato è maggiore di 0, allora il numero di punti condizionali del programma sono un numero finito.

QM-PROD-12 Complessità della classe (CCLA)

- **Descrizione:** la metrica CCLA misura il quantitativo medio del numero di metodi presenti nelle classi e permette di comprendere la complessità generale delle stesse;
- **Unità di misura:** la metrica è espressa tramite un numero decimale;
- **Formula:** la formula della metrica è la seguente:

$$CCLA = \frac{\text{numero totale di metodi}}{\text{numero totale di classi}}$$

- **Risultato:** il risultato della formula ha i seguenti significati:
  - se il risultato è pari a 0, allora le classi non hanno metodi;
  - se il risultato è maggiore di 0, allora in media una classe della componente software ha un numero finito di metodi.

QM-PROD-13 Numero di code smell rilevati (NCS)

- **Descrizione:** la metrica viene usata per rilevare il numero di CODE SMELL$_G$ attraverso SONARCLOUD$_G$; un CODE SMELL$_G$ fa riferimento ad una best practice mancata che i programmatori dovrebbero seguire nel corso dello sviluppo per rendere il codice leggibile, privo di errori e manutenibile;
- **Unità di misura:** la metrica è espressa tramite un numero intero;

- **Fonte:** il dato viene recuperato attraverso le statistiche di SONARCLOUD$_G$;
- **Risultato:** il risultato rilevato ha i seguenti significati:
  - se il risultato è pari a 0, allora il codice scritto ha rispettato tutte le *best practice* segnalate;
  - se il risultato è maggiore di 0, allora il codice scritto non ha rispettato tutte le *best practice* segnalate.

QM-PROD-14 Tempo di risoluzione code smell (TCS)

- **Descrizione:** la metrica viene usata per rilevare il tempo stimato di risoluzione dei CODE SMELL$_G$ attraverso SONARCLOUD$_G$; il tempo riportato risulta essere una stima che viene calcolata direttamente da SONARCLOUD$_G$ sulla base della quantità di CODE SMELL$_G$ e della complessità di risoluzione di ognuno di essi;
- **Unità di misura:** la metrica è espressa in minuti;
- **Fonte:** il dato viene recuperato attraverso le statistiche di SONARCLOUD$_G$;
- **Risultato:** il risultato rilevato ha i seguenti significati:
  - se il risultato è pari a 0, allora non sono stati trovati CODE SMELL$_G$ da risolvere;
  - se il risultato è maggiore di 0, allora sarà necessario il quantitativo di minuti indicato per risolvere tutti i CODE SMELL$_G$ rilevati.

QM-PROD-15 Percentuale di duplicazione del codice (DUPC)

- **Descrizione:** la metrica viene usata per rilevare la percentuale di codice duplicato nel corso dello sviluppo;
- **Unità di misura:** la metrica è espressa in percentuale (%);
- **Fonte:** il dato viene recuperato attraverso le statistiche di SONARCLOUD$_G$;
- **Risultato:** il risultato rilevato ha i seguenti significati:
  - se il risultato è pari a 0, allora il codice scritto non è stato duplicato su più parti del programma;
  - se il risultato è maggiore di 0, allora una parte del codice scritto è stato duplicato su più punti del programma.

QM-PROD-16 Numero di violazioni dello standard di codifica (NVSC)

- **Descrizione:** la metrica viene usata per rilevare il numero di violazioni dello standard di codifica;
- **Unità di misura:** la metrica è espressa in percentuale (%);
- **Fonte:** il dato viene recuperato attraverso i report di *Maven* durante la compilazione e tramite GITHUB ACTIONS$_G$;
- **Risultato:** il risultato rilevato ha i seguenti significati:
  - se il risultato è pari a 0, allora il codice scritto ha rispetto a pieno lo standard di codifica;
  - se il risultato è maggiore di 0, allora tutto o una parte del codice scritto non ha rispettato lo standard di codifica.

2.2.5.6 QP-3 Sviluppo

Scopo

Durante lo sviluppo, si vuole monitorare la capacità del prodotto di soddisfare tutti i requisiti richiesti dal proponente per determinare il grado di completamento.

Introduzione alle Metriche

Per il processo di sviluppo si farà uso delle seguenti metriche:

- QM-PROC-7 Satisfied mandatory requirements (SMR);
- QM-PROC-8 Satisfied desirable requirements (SDR);
- QM-PROC-9 Satisfied optional requirements (SOR);
- QM-PROC-10 Numero di commits (NCOM).

QM-PROC-7 Satisfied mandatory requirements (SMR)

- **Descrizione:** la metrica SMR indica il quantitativo di requisiti obbligatori soddisfatti (progettati, sviluppati e verificati) fino alla data corrente; questa metrica permette sia al gruppo, che al committente, di comprendere la percentuale di completezza base del prodotto;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{SMR} = \frac{\text{requisiti obbligatori soddisfatti}}{\text{requisiti obbligatori totali}} \times 100$$

- **Risultato:**
  - un risultato pari a 0% indica che non è stato soddisfatto ancora alcun requisito obbligatorio;
  - un risultato superiore allo 0% ma inferiore al 100% indica che alcuni requisiti obbligatori sono stati soddisfatti;
  - un risultato pari a 100% indica che sono stati soddisfatti tutti i requisiti obbligatori.

QM-PROC-8 Satisfied desirable requirements (SDR)

- **Descrizione:** la metrica SDR indica il quantitativo di requisiti desiderabili soddisfatti (progettati, sviluppati e verificati) fino alla data corrente; questa metrica permette sia al gruppo, che al committente, di comprendere la percentuale di valore aggiunto del prodotto;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{SDR} = \frac{\text{requisiti desiderabili soddisfatti}}{\text{requisiti desiderabili totali}} \times 100$$

- **Risultato:**
  - un risultato pari a 0% indica che non è stato soddisfatto ancora alcun requisito desiderabile;
  - un risultato superiore allo 0% ma inferiore al 100% indica che alcuni requisiti desiderabili sono stati soddisfatti;
  - un risultato pari a 100% indica che sono stati soddisfatti tutti i requisiti desiderabili.

QM-PROC-9 Satisfied optional requirements (SOR)

- **Descrizione:** la metrica SDR indica il quantitativo di requisiti desiderabili soddisfatti (progettati, sviluppati e verificati) fino alla data corrente;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{SOR} = \frac{\text{requisiti opzionali soddisfatti}}{\text{requisiti opzionali totali}} \times 100$$

- **Risultato:**
  - un risultato pari a 0% indica che non è stato soddisfatto ancora alcun requisito opzionale;
  - un risultato superiore allo 0% ma inferiore al 100% indica che alcuni requisiti opzionali sono stati soddisfatti;
  - un risultato pari a 100% indica che sono stati soddisfatti tutti i requisiti opzionali.

QM-PROC-10 Numero di commit (NCOM)

- **Descrizione:** la metrica mostra il numero di commit che sono stati inseriti nel repository nell'arco di tempo definito;
- **Unità di misura:** la metrica viene espressa con un numero intero;
- **Fonte:** il dato viene recuperato attraverso le statistiche di GITHUB$_G$;
- **Risultato:**
  - un risultato pari a 0 indica che non ci sono stati commit e pertanto il gruppo non ha lavorato;
  - un risultato superiore a 0 indica che il gruppo ha lavorato attivamente al progetto.

2.2.6 Strumenti

Di seguito sono elencati gli strumenti che verranno utilizzati nel processo di sviluppo.

2.2.6.1 IntelliJ Idea - Ultimate Edition

IntelliJ Idea è un IDE sviluppato da *JetBrains* che viene utilizzato per lo sviluppo di applicativi JAVA$_G$ e permette di agevolare ampiamente lo sviluppatore, grazie agli strumenti aggiuntivi che possono essere facilmente configurati.
Include anche la possibilità di usare un debugger, eseguire refactoring del codice, configurare un sistema di versionamento e integrarsi con i framework più utilizzati per JAVA$_G$, come SPRING$_G$.
Il software è a pagamento, ma gratuito per gli studenti universitari.

- IntelliJ Idea genera una cartella `.idea` che non deve essere aggiunta al sistema di versionamento;
- analogamente, anche le cartelle di output, `out/` e `target/`, non devono essere versionate.

2.2.6.2 PHPStorm

PHPStorm è un IDE sviluppato da *JetBrains*, molto simile a IntelliJ Idea, che viene usato per lo sviluppo della web app, la quale fa uso di HTML, PHP e JAVASCRIPT$_G$.
Si integra facilmente con LARAVEL$_G$ e altri framework molto usati, in particolare permette di configurare tutto in modo automatico, agevolando lo sviluppatore.
PHPStorm è gratuito per gli studenti universitari e può essere riscattato, allo stesso modo di IntelliJ Idea, dal sito ufficiale.

- PHPStorm genera una cartella `.idea` che non deve essere aggiunta al sistema di versionamento;
- analogamente, anche le cartelle di output, `out/` e `target/`, non devono essere versionate.

2.2.6.3 WebStorm

WebStorm è un IDE sviluppato da *JetBrains*, molto simile a PHPStorm, che viene usato per lo sviluppo del BOT TELEGRAM$_G$, il quale fa uso di JAVASCRIPT$_G$.
Si integra facilmente con NODE.JS$_G$ e altri framework molto usati, in particolare permette di configurare tutto in modo automatico, agevolando lo sviluppatore.
WebStorm è gratuito per gli studenti universitari e può essere riscattato, allo stesso modo di PHPStorm, dal sito ufficiale.

- WebStorm genera una cartella `.idea` che non deve essere aggiunta al sistema di versionamento;
- analogamente, anche le cartelle di output, `out/` e `target/`, non devono essere versionate.

2.2.6.4 Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da *Microsoft* per Windows, Linux e MacOS; esso include una serie di estensioni che possono essere utilizzate al fine di arricchire l'editor in base alle necessità.
Nell'ambito del progetto, Visual Studio Code può essere utilizzato per lo sviluppo del codice sorgente per board come ARDUINO UNO$_G$ e ESP8266$_G$, con le quali è possibile simulare fisicamente un dispositivo industriale.

2.2.6.5 Portainer

Portainer è un applicativo web usato per la gestione dei CONTAINER$_G$, istanziati in una macchina locale o remota, che fa uso di DOCKER$_G$.
L'applicativo stesso risulta essere una istanza di un CONTAINER$_G$, che può essere installato in qualunque momento. Dal pannello di gestione è possibile:

- aggiungere, modificare e rimuovere container;
- avviare, fermare e riavviare container;
- visualizzare i logs di un singolo container;
- operare dentro il container da riga di comando;
- modificare la configurazione di rete e di avvio del container;
- aggiungere ed eseguire il processo di build delle IMMAGINI$_G$ di DOCKER$_G$.

2.2.6.6 Kafka Manager

Kafka manager è una web app che permette di monitorare e operare sugli stream di KAFKA$_G$.
La web app è facilmente accessibile tramite browser e permette di gestire nel modo più completo ciò che avviene sulle singole istanze di TOPIC$_G$, verificando effettivamente il funzionamento dell'infrastruttura. Inoltre, è possibile:

- creare manualmente nuovi topic;
- cancellare topic esistenti;
- aggiornare dei parametri di configurazione di un particolare topic;
- controllare il flusso di produzione e consumazione dei messaggi.

2.2.6.7 Draw.io

Draw.io è un'applicazione online gratuita e collaborativa, utilizzabile nella suite di GOOGLE DRIVE$_G$, per la realizzazione di diagrammi di molteplici tipologie.
In particolare, risulta molto utile per la rappresentazione di diagrammi UML di qualsiasi tipologia, grazie ad un insieme di forme e figure in linguaggio UML predefinite all'interno dell'applicazione stessa e facilmente componibili per la costruzione dei diagrammi.

2.2.6.8 Firefox

Browser web sviluppato dalla *Mozilla Foundations* che viene utilizzato per visualizzare la web app e accedere alle interfacce online di DOCKER$_G$.
Il browser è gratuito e scaricabile dal sito ufficiale per qualunque sistema operativo.

2.2.6.9 Chrome

Chrome è un browser web sviluppato da *Google* che viene utilizzato in alternativa a *Firefox*, per la visualizzazione della web app e per accedere agli strumenti online.
Viene utilizzato sia a livello desktop, che a livello mobile, specialmente negli smartphone *Android*.

2.2.6.10 Safari

Safari è un browser web sviluppato da *Apple* che viene utilizzato per accedere e testare la web app su *MacOs*.
Esiste una versione mobile del browser che viene utilizzata nativamente negli smartphone *Apple*.

3 Processi di supporto

3.1 Documentazione

3.1.1 Scopo

Lo scopo principale di questo capitolo è fornire una guida esaustiva di tutti gli standard e regole per quanto riguarda la stesura ed approvazione dei documenti.

3.1.2 Aspettative

Si vuole fornire uno strumento per la stesura dei documenti unico per tutto il gruppo in modo da avere una documentazione uniforme e aderente agli standard e regole sotto riportate.

3.1.3 Descrizione

Questo capitolo fornisce i dettagli su come deve essere redatta e verificata la documentazione. Tutte le norme descritte devono essere rispettate in pieno da tutti i documenti, sia interni che esterni, rilasciati durante il ciclo di vita del software.

3.1.4 Attività

3.1.4.1 Implementazione del processo

Ciclo di vita

Il ciclo di vita dei documenti è suddiviso in varie attività, eventualmente ripetibili:

- **stesura:** è il processo di scrittura del documento in sé; questa attività viene assegnata ad un redattore che, terminato il suo svolgimento, farà riferimento al responsabile, il quale autorizzerà l'avanzamento del documento all'attività successiva;
- **verifica:** è l'attività eseguita dai verificatori, i quali hanno il compito di controllare che la stesura del documento sia avvenuta in modo corretto, sintatticamente e semanticamente, seguendo le norme di progetto. Ogni problema viene riferito al responsabile che provvederà a notificare il redattore e riporterà il documento all'attività di stesura. Quando questa attività sarà portata a termine con successo il responsabile farà avanzare il documento nell'ultima attività del ciclo di vita;
- **approvazione:** è l'ultima attività del ciclo di vita del documento, in cui il verificatore ha terminato il suo compito ed ha comunicato l'esito positivo al responsabile. Il responsabile procederà a confermare il documento ed eseguire il rilascio.

Documenti ufficiosi

I documenti ufficiosi sono utilizzati all'interno dell'ambiente di lavoro ed sono divisi in due categorie:

- **informativi:** hanno il solo scopo di passare informazioni meno rilevanti tra i membri del gruppo (es. appunti, richieste, riflessioni);
- **proto-ufficiali:** sono tutti i documenti che in futuro diventeranno "ufficiali" ma sono in attesa di revisione e verifica.

Documenti ufficiali

Si dicono ufficiali i documenti che:

- sono stati revisionati, verificati ed approvati dal responsabile di progetto;
- sono gli unici rilasciabili all'esterno del gruppo di progetto.

I documenti che verranno prodotti sono:

Studio di fattibilità

Lo *studio di fattibilità* ha l'obiettivo di descrivere brevemente ciò che ogni CAPITOLATO$_G$ ha da proporre, elencando gli aspetti che il gruppo ha individuato come i più interessanti e le loro criticità. I destinatari di questo documento sono i membri del gruppo Red Round Robin.

Piano di progetto

Lo scopo del *piano di progetto* è organizzare le attività, analizzandone i rischi, suddividendole ed assegnandole ai vari membri del gruppo, in modo tale da raggiungere i risultati nel modo più efficace ed efficiente possibile. I destinatari di questo documento sono il committente ed il proponente oltre al gruppo Red Round Robin.

Piano di qualifica

Il *piano di qualifica* ha lo scopo di presentare i metodi di verifica e validazione adottate dal gruppo Red Round Robin per garantire la qualità di prodotto e di processo. Per assicurarci di ciò, viene applicato un sistema di verifica continua che permette l'individuazione di eventuali errori per poterli risolvere efficacemente, limitando gli sprechi di tempo. I destinatari di questo documento sono il proponente ed il committente, oltre al gruppo Red Round Robin.

Analisi dei requisiti

Lo scopo dell'*analisi dei requisiti* è la candidatura del gruppo Red Round Robin allo svolgimento del progetto relativo al capitolato C6 - ThiReMa. All'interno di questa analisi è possibile seguire la classificazione, il tracciamento e la descrizione dettagliata dei requisiti individuati dall'analisi del capitolato scelto. I destinatari di questo documento sono i membri del gruppo Red Round Robin, il committente e il proponente.

Verbali

I verbali hanno lo scopo di riassumere, in modo coinciso e preciso, tutti gli argomenti che sono stati discussi in una riunione, sia interna che esterna. È prevista un'unica stesura del verbale per ogni riunione in quanto una modifica ad una decisione avrebbe un effetto retroattivo.

Glossario

Il glossario ha la funzione di disambiguare alcune parole all'interno di determinati contesti. I destinatari del documento sono i membri del gruppo Red Round Robin, il committente e il proponente.

3.1.4.2 Sviluppo e design

Template LaTeX

Si è deciso di utilizzare una struttura template LaTeX per facilitare il versionamento e la stesura dei documenti. Inoltre, l'utilizzo di tale struttura fornisce uniformità al layout di tutti i documenti.

Struttura generale dei documenti

Un file "main.tex" provvederà a raccogliere tutte le sezioni, pacchetti e comandi necessari per la sua compilazione. Tutti i documenti hanno una struttura predefinita e determinata.

Frontespizio

Il frontespizio ha la funzione di fornire i dati principali del documento. Esso presenterà il logo e il nome del gruppo, il titolo del documento e la sua appartenenza ad un determinato progetto, le informazioni sul documento quali:

- **versione:** versione attuale del documento;
- **uso:** destinazione d'uso del documento, che potrà essere "interno" o "esterno";
- **stato:** attuale stato del documento, che potrà essere "in redazione" o "approvato";
- **destinatari:** destinatari del documento;
- **redattori:** lista dei membri del gruppo che si sono occupati della stesura dello specifico documento;
- **verificatori:** lista dei membri del gruppo che si sono occupati della verifica dello specifico documento;
- **approvazione:** nominativo del membro del gruppo che ha approvato il documento per il rilascio.

Tutti gli elementi di questa pagina sono centrati ed incolonnati.

Registro modifiche

Il registro delle modifiche occupa la seconda pagina del documento e consiste in una tabella contenente le informazioni riguardanti il ciclo di vita del documento. Più precisamente, la tabella riporta per ogni modifica:

- **versione:** versione del documento relativa alla modifica effettuata;
- **descrizione:** breve descrizione della modifica effettuata;
- **data:** data in cui la modifica è stata effettuata;
- **autore:** nominativo della persona che ha effettuato la modifica;
- **ruolo:** ruolo della persona che ha effettuato la modifica.

Indice

L'indice ha lo scopo di riepilogare e dare una visione generale della struttura del documento, mostrando le parti di cui è composto. L'indice ha una struttura standard: numero e titolo del capitolo, con eventuali sottosezioni, e il numero della pagina del contenuto; inoltre, ogni titolo è un link alla pagina del contenuto. L'indice dei contenuti è seguito da un eventuale indice per le tabelle e le figure presenti nel documento.

Contenuto principale

La struttura del contenuto principale di una pagina è così composta:

- in alto a sinistra è presente il logo del gruppo;
- in alto a destra è riportata la sezione alla quale la pagina appartiene;
- il contenuto principale è posto tra l'intestazione e il piè di pagina;
- una riga divide il contenuto principale e il piè di pagina;
- in basso a sinistra è presente il nome del documenti con relativa versione;
- in basso a destra riporta il numero della pagina attuale ed il numero totale delle pagine che compongono il documento; se la pagina non fa parte del contenuto principale la sua numerazione avviene tramite i numeri romani.

Verbali

I verbali seguono la stessa struttura di tutti gli altri documenti ad eccezione fatta della numerazione delle pagine, che usa la notazione romana anziché araba. Inoltre, il verbale è suddiviso in:

- **introduzione:** essa contiene:
  - **luogo:** luogo o la piattaforma online, in caso di incontri virtuali;
  - **data:** data dell'incontro;

  - **ora di inizio:** l'ora dell'inizio dell'incontro in formato HH:MM;
  - **ora di fine:** l'ora della fine dell'incontro in formato HH:MM;
  - **ordine del giorno:** consiste in una lista degli argomenti che il gruppo si è proposto di discutere durante l'incontro;
  - **presenze:** contiene il numero totale dei partecipanti, la lista dei presenti e la lista degli assenti con eventuale giustifica;
  - **segretario:** scelto tra i componenti del gruppo e incaricato di prendere nota delle discussioni effettuate; stenderà il verbale della riunione;

- **svolgimento:** per ogni punto presente nell'ordine del giorno, viene riportato un riassunto di ciò che è stato trattato durante l'incontro;
- **tracciamento delle decisioni:** è un riepilogo in formato tabellare delle decisioni prese durante l'incontro; esso è composto da:
  - **codice:** del tipo "VT_AAAA-MM-GG_X.Y" dove la prima lettera indica che il documento è un verbale(V), la seconda indica la sua tipologia, esterno(E) o interno(I), seguito dalla data dell'incontro a cui fa riferimento e terminato, infine, da un numero che indica il numero del verbale(X) ed un secondo numero che indica il punto all'ordine del giorno a cui si riferisce(Y);
  - **descrizione:** breve descrizione riassuntiva della decisione presa riguardante il punto dell'ordine del giorno.

Ogni verbale dovrà essere denominato seguendo il formato "Verbale riunione #X", dove la "X" corrisponde al numero del verbale in ordine temporale.

Glossario

All'interno del glossario saranno presenti tutte parole con le seguenti caratteristiche:

- sono presenti in almeno un documento;
- trattano argomenti di natura tecnica;
- trattano argomenti ambigui e/o poco conosciuti;
- rappresentano delle sigle e/o degli acronimi non di uso comune.

Inoltre, il glossario è strutturato in maniera precisa seguendo due regole:

- i termini seguono l'ordine lessicografico;
- ogni termine è spiegato in maniera chiara e in nessun modo ambigua.

La stesura del glossario deve avvenire in parallelo alla stesura dei documenti al fine di evitare confusione tra i termini. Inoltre, ogni parola dei documenti riportata nel glossario, deve essere caratterizzata dallo stile "maiuscoletto" con il pedice "G". Non si richiede tale stile se la parola presente nel documento è stata precedentemente caratterizzata con la suddetta notazione.

Lettere

La lettera di presentazione dovrà seguire il classico layout per lettere, il che implica la presenza dei mittenti e destinatari, il logo del gruppo e la lista di tutti i documenti rilasciati, nonché il preventivo per il progetto.

Norme tipografiche

Convenzioni sui nomi dei file

si è deciso di usare la convenzione CAMEL CASE$_G$ per i nomi di file e cartelle. Le regole seguite saranno le seguenti:

- il nome dei file composti da più parole avranno la prima lettera minuscola ed ogni parola in seguito inizierà con una maiuscola;
- tra le parole non sarà presente alcun separatore;
- le preposizioni non verranno omesse;
- sono escluse da questa sintassi le estensione dei file.

Alcuni esempi corretti sono:

- convenzioniSuiNomiDeiFile.tex;
- documentazione.tex.

Alcuni esempi non corretti sono:

- ConvenzioniSuiNomiDeiFile (la prima lettera è maiuscola);
- convenzioni Sui Nomi Dei File (usa un carattere separatore);
- convenzioni_Sui_Nomi_Dei_File (usa un carattere separatore);
- convenzioniNomiFile (omette preposizioni).

Stile del testo

I vari stili del testo hanno una specifica funzione semantica:

- **corsivo:** viene utilizzato per denotare termini tecnici appartenenti ad una particolare tecnologia, esempio *branch*;
- **grassetto:** viene utilizzato per evidenziare le parole con la definizione della stessa in seguito, per esempio in un elenco puntato, che includeranno i due punti in grassetto, per esempio "**def.:** abbreviazione per la parola definizione"; oppure per denotare le sezioni e sotto-sezioni dei documenti;
- **maiuscoletto:** viene utilizzato per denotare parole che sono:
  - riferimenti a documenti esterni, con pedice una "D";
  - appartenenti al glossario, con pedice una "G".

Elenchi puntati

Ogni elemento dell'elenco deve essere seguito da un punto e virgola, fatta eccezione per l'ultimo elemento che sarà seguito da un punto; di conseguenza, la prima lettera di ogni voce dell'elenco deve essere minuscola, ad eccezione della prima frase se quest'ultima è posta all'inizio di paragrafo. Gli elenchi avranno punto elenco differente a seconda della loro tipologia:

- per gli elenchi non ordinati si è scelto di usare come punto elenco un cerchietto pieno e come sub-punto elenco il trattino;
- per gli elenchi ordinati si è optato per un punto elenco "flessibile", ossia possono essere usati sia i numeri che i letterali, purché quest'ultimi siano in minuscolo, seguiti da un punto, esempio "1." o "a.".

Formati comuni

- **data:** viene utilizzato lo standard ISO 8601, esempio 2020-01-22, per la rappresentazione di tutte le date presenti della documentazione;
- **ora:** viene utilizzato il formato HH:MM;
- **versione:** viene utilizzato il formato vXX.YY.ZZ+bJJ.KK.

Sigle

Il progetto richiede la redazione di un insieme di documenti; segue l'elenco dei documenti con relative sigle:

- **documenti esterni:**
  - **analisi dei requisiti - AdR:** descrive le caratteristiche del software;
  - **piano di progetto - PdP:** riguarda la gestione del progetto, ossia descrive parametri come fattibilità, costi, vincoli del progetto;
  - **piano di qualifica - PdQ:** descrive la qualità del software e dei processi coinvolti, come e con quali strumenti si intende raggiungere tale qualità;
  - **manuale utente - MU:** manuale per gli utilizzatori del software;
  - **manuale sviluppatore - MS:** manuale per gli sviluppatori e manutentori;
- **documenti interni:**
  - **glossario - G:** raccoglie tutti i termini che necessitano di una disambiguazione e/o una descrizione più approfondita;
  - **norme di progetto - NdP:** è una raccolta di tutte le regole e le norme utilizzate durante il ciclo di vita del software;
  - **studio di fattibilità - SdF:** descrive i vari capitolati analizzando brevemente i loro pro e contro;
- **verbali - V:** essi possono essere sia interni che esterni e descrivono in maniera concisa tutti gli argomenti discussi e le decisione prese durante un incontro.

Inoltre, il ciclo di vita del progetto è diviso in quattro fasi:

- **revisione dei requisiti - RR:** studio iniziale del capitolato;
- **revisione di progettazione - RP:** definizione dell'architettura e della fattibilità del software;
- **revisione di qualifica - RQ:** produzione di codice e descrizione dettagliata delle sue componenti;
- **revisione di accettazione - RA:** approvazione del prodotto da parte del cliente e rilascio del software.

Altre sigle utilizzate all'interno dei documenti sono:

- AWS: AMAZON WEB SERVICE$_G$;
- ETH: ethereum;
- EVM: ethereum virtual machine;
- LR: REGRESSIONE LINEARE$_G$;
- SVM: SUPPORT VECTOR MACHINE$_G$;
- ML: MACHINE LEARNING$_G$;
- JS: JAVASCRIPT$_G$;
- AoA: angolo di arrivo;
- UI: USER INTERFACE$_G$;
- PaaS: platform as a service;
- IaaS: infrastructure as a service;
- FaaS: function as a service;
- CaaS: container as a service;
- BaaS: back end as a service;
- IoT: internet of things;
- DB: DATABASE$_G$;
- VCS: VERSION CONTROL SYSTEM$_G$.

Elementi grafici

Tabelle

Le tabelle sono sempre accompagnate da un titolo e dal numero della tabella; sono indicizzate separatamente rispetto al resto del contenuto.

Immagini

Le immagini sono sempre accompagnate da una didascalia descrittiva e dal numero della figura; sono indicizzate separatamente rispetto al resto del contenuto.

Diagrammi UML

I diagrammi UML vengono inseriti all'interno della documentazione sotto forma di immagini.

3.1.5 Metriche

3.1.5.1 QC-1 Comprensione

Scopo

Per fornire una documentazione fruibile e comprensibile si è deciso di monitorare la sua qualità tramite delle metriche significative.

Introduzione alle metriche di qualità

Per la comprensione si è deciso di utilizzare le seguenti metriche:

- QM-PROD-1 Indice di Gulpease$_G$ (GULP);
- QM-PROD-2 Correttezza ortografica (CORT);
- QM-PROD-3 Limite lunghezza video tutorial (LLVT).

QM-PROD-1 Indice di Gulpease (GULP)

- **Descrizione:** la metrica GULP permette di misurare la leggibilità di un documento basandosi sulla formula riportata di seguito;
- **Unità di misura:** la metrica è espressa tramite un numero intero;
- **Formula:** la formula della metrica è la seguente: $GULP = 89 + \frac{300 \times \#frasi - 10 \times \#lettere}{\#parole}$
- **Risultato:** il risultato della formula ha i seguenti significati:
  - se il risultato è pari 0 allora il documento non esiste e/o la sua leggibilità è terribile;
  - se il risultato è maggiore di 40 allora il documento esiste ed è leggibile da chi possiede un diploma superiore;
  - se il risultato è maggiore di 60 allora il documento esiste ed è leggibile da chi possiede una licenza media;
  - se il risultato è maggiore di 80 allora il documento esiste ed è leggibile da chi possiede una licenza elementare;
  - se il risultato è pari a 100 allora il documento esiste ed è molto più che leggibile.

QM-PROD-2 Correttezza ortografica (CORT)

- **Descrizione:** la metrica CORT permette di misurare la correttezza, a livello lessicografico, di un documento;
- **Unità di misura:** la metrica è espressa tramite un numero intero;
- **Formula:** la formula della metrica è la seguente: *CORT = # numero di errori ortografici*
- **Risultato:** il risultato della formula ha i seguenti significati:
  - se il risultato è pari 0 allora il documento non esiste o non ha errori ortografici;
  - se il risultato è maggiore di 0 allora il documento esiste e presenta errori ortografici.

QM-PROD-3 Limite lunghezza video tutorial (LLVT)

- **Descrizione:** la metrica permette di misurare la lunghezza dei video tutorial realizzati per il manuale utente; si vuole cercare di contenere il tempo richiesto per illustrare una funzionalità, riducendo eventuali spiegazioni superflue e tenendo sotto controllo il tempo richiesto per la realizzazione;
- **Unità di misura:** la metrica è espressa in secondi;
- **Risultato:** il risultato viene confrontato con una soglia massima e minima e ha i seguenti significati:
  - se il risultato è sotto la soglia minima, allora la funzionalità illustrata è stata spiegata troppo velocemente;
  - se il risultato è sopra la soglia massima, allora la funzionalità illustrata è stata spiegata troppo lentamente;
  - se il risultato è contenuto tra la soglia minima e la soglia massima, allora la funzionalità illustrata è stata spiegata con un tempo adeguato;

3.1.6 Strumenti

3.1.6.1 LaTeX

Per la scrittura dei documenti è stato scelto di usare LaTeX, linguaggio di markup basato sul programma di tipografia digitale TeX. Questo permette di scrivere documenti in maniera modulare e collaborativa.

3.1.6.2 Editor di testo

Il gruppo ha deliberato su un utilizzo eterogeneo degli editor di testo, poiché ogni membro ha familiarità con editor ed ambienti differenti.

3.1.6.3 Diagrammi

I software utilizzati per la realizzazione dei diagrammi sono i seguenti:

- **Draw.io:** applicazione online gratuita e collaborativa utilizzabile nella suite di GOOGLE DRIVE$_G$, per la realizzazione di casi d'uso, diagrammi ER e schemi semplici;
- **Gantt Project:** applicazione open source per la realizzazione di diagrammi di Gantt che mostrano lo svolgimento delle attività.

3.1.6.4 Google Drive

Si utilizza *Google Drive* per salvare file utili di qualunque genere e per realizzare presentazioni powerpoint, collaborando direttamente online, tramite la relativa suite.

3.2 Gestione della configurazione

3.2.1 Scopo

La gestione della configurazione definisce i principi normativi utili a predisporre il WORKSPACE$_G$ per tutto il gruppo, semplificando e automatizzando la conservazione dei documenti e delle componenti software, che andranno a formare il prodotto finale.

3.2.2 Aspettative

La gestione della configurazione si predispone a rendere usabile l'ambiente di lavoro, controllando attivamente e autonomamente tutte le attività in modo metodico e organizzato. Ciò è importante perché riduce la complessità di coordinamento nel corso dello sviluppo del prodotto, sia che riguardi la documentazione, sia che riguardi le componenti software.

3.2.3 Descrizione

Il processo di gestione della configurazione si compone di una serie di attività e strumenti atte a costituire un ambiente di lavoro semplice e ordinato. In particolare, si fa uso dei seguenti strumenti:

- repository;
- tecnologie scelte.

Gli strumenti vengono utilizzati per svolgere le seguenti attività:

- versionamento e rilascio del prodotto;
- versionamento e rilascio dei documenti;
- versionamento e rilascio dei componenti software;
- continuous integration;
- notification & monitoring;

3.2.4 Attività

3.2.4.1 Versionamento e rilascio del prodotto

Il prodotto viene inteso come l'insieme di componenti che dovranno essere progettate, sviluppate, verificate e validate prima di essere consegnate al cliente. Il prodotto si compone principalmente di:

- documentazione;
- componenti software.

Ciascuna componente viene versionata seguendo la propria evoluzione, mentre nel caso del prodotto si segue un versionamento più macroscopico e basato sulle BASELINE$_G$. Nell'ambito del progetto, il gruppo ha deciso di integrare una prassi di versionamento che permetta di avere un riferimento del prodotto dalle singole componenti sviluppate.

Sintassi codice di versionamento

Al numero di versione di un prodotto associato ad una baseline si aggiunge il seguente identificativo:

$$+\mathrm{b}[\alpha].[\beta]$$

- (α): numero identificativo del rilascio del prodotto
  - parte da 0 e non si resetta mai;
  - viene incrementato quando tutte le componenti sono state concluse e pronte per la consegna al cliente;
- (β): numero identificativo che viene associato a una baseline del prodotto:
  - parte da 0 e si resetta quando α viene incrementato;
  - viene incrementato quando si raggiunge una nuova baseline di prodotto.

Esempi di versionamento del prodotto

- **v0.5.1+b0.1**: indica la versione 0.5.1 di una componente proveniente dalla baseline di prodotto in versione 0.1;
- **v1.13.1+b0.2**: indica la versione 1.12.1 di una componente proveniente dalla baseline di prodotto in versione 0.2;
- **v3.0.2+b1.3**: indica la versione 3.0.2 di una componente proveniente dalla baseline di prodotto in versione 1.3.

Rilascio

Il rilascio del prodotto viene effettuato nel momento in cui la prima cifra della versione viene incrementata. Ad essa si devono allineare tutte le altre componenti con delle versione approvate ed autorizzate per il rilascio.

3.2.4.2 Versionamento e rilascio dei documenti

Tutti i file che riguardano la documentazione vengono conservati in un repository e ogni documento viene versionato per mezzo di un identificativo, in base alla sua fase di avanzamento. Questo permette di poter fare riferimento alle nuove versioni del documento durante tutto il ciclo di vita del software.

Sintassi codice di versionamento

Ciascun documento possiede un identificativo di versionamento su ogni pagina che rispetta il seguente formalismo:

$$\mathbf{v}[\alpha].[\beta].[\gamma]$$

- (α): numero identificativo che rappresenta un insieme di modifiche sostanziali della struttura e dei contenuti delle sezioni del documento;
  - parte da 0 e non si resetta mai;
  - viene incrementato solo dal responsabile, o da una procedura automatica da lui approvata, a seguito di un accertamento di importanti cambiamenti nel documento, quali:
    - aggiunta o modifica di almeno un quarto del numero di sezioni del documento;
    - cambiamento radicale della struttura delle sezioni;
    - aggiunta di una nuova sezione di primo livello o di più appendici;
- (β): numero identificativo che rappresenta un accertamento e un'approvazione del documento a seguito di modifiche minori;
  - parte da 0 e si resetta solo a incrementi di α;
  - viene incrementato solo dal responsabile, o da una procedura automatica da lui approvata, a seguito del raggiungimento di una baseline, se il prodotto è conforme;
  - le modifiche minori possono riguardare:
    - la modifica di un numero ristretto di sezioni (es: una o due sezioni);
    - l'aggiunta di una nuova sezione di secondo livello o di un'appendice;
- (γ): numero rappresentativo di una aggiunta o modifica verificata fatta al documento, in modo singolare
  - parte da 0 e si resetta solo a incrementi di β o di α;
  - viene incrementato da un revisore dopo aver verificato che una sezione sia conforme a quanto deciso.

Generalmente, ogni documento prima di poter essere rilasciato verso l'esterno dovrà essere stato approvato per rilasciato.

Esempi codice di versionamento dei documenti

- `v0.2.1`;
- `v1.12.1`;
- `v3.0.2`.

Gestione delle modifiche

Le modifiche ai documenti vengono gestite seguendo il **registro delle modifiche** presente in tutti i documenti nella pagina successiva al frontespizio. All'interno di questo documento si tiene traccia di ogni aggiunta, modifica, eliminazione, revisione o approvazione da parte di tutti i membri del team. La normativa sul formalismo può essere trovata al paragrafo 3.1.4.2 a pagina 40.

Rilascio

Un documento viene rilasciato alle parti proponenti solamente quando vi è un incremento del primo numero (α), che ne implica una approvazione da parte del responsabile.
Per quanto concerne la distribuzione interna, tutti gli ARTEFATTI$_G$ dei documenti realizzati durante la fase di sviluppo sono resi disponibili in modo rapido e automatizzato a tutti i membri del gruppo, a cui vengono notificate tutte le modifiche tramite il WORKSPACE$_G$ di *Slack*.

Integrazione con il versionamento di prodotto

La documentazione viene interpretata come componente, e in quanto tale riceve come aggiunta al proprio codice di versionamento anche l'identificativo di prodotto. Questo identificativo aggiuntivo viene esplicitato solamente all'interno del documento e non nella denominazione dei file.

3.2.4.3 Versionamento e rilascio dei componenti software

I sorgenti del software che riguardano la codifica e la configurazione del prodotto da realizzare sono mantenuti nel repository, insieme alla documentazione. Ogni file viene versionato con un apposito storico delle modifiche, mentre ogni componente software viene versionato come BASELINE$_G$ di prodotto in relazione alle funzionalità presenti e dei requisiti obbligatori implementati.

Sintassi codice di versionamento

Il versionamento del software viene eseguito sulla base delle implementazioni effettuate a livello di codifica. In aggiunta, si definisce una sintassi di stato nella versione per definirne l'usabilità.

$$\text{v}[\delta].[\epsilon].[\mu]\text{-}[\lambda]$$

- (δ): numero identificativo del rilascio software per una MAJOR RELEASE$_G$
  - parte da 0 e non si resetta mai;
  - viene incrementato solo dopo aver implementato tutti i requisiti obbligatori;
- (ϵ): numero identificativo per una MINOR RELEASE$_G$
  - parte da 0 e si resetta solo a incrementi di δ;
  - viene incrementato solo dopo l'implementazione di uno o più requisiti;
- (μ): numero rappresentativo per una PATCH$_G$
  - parte da 0 e si resetta solo a incrementi di ϵ o di δ;
  - viene incrementato a ogni modifica di uno o più requisiti e ad ogni cambio di configurazione del software;
- (λ): sigla che identifica uno stato del software; può avere i seguenti significati in ordine crescente di stato
  - `dev`: derivante da **dev**elopment, versione ancora in sviluppo
    - software non completo, che ha ricevuto aggiunte, modifiche o eliminazioni recenti;

    - passa tutti i test sviluppati fino a quel momento;
    - **non** si presta all'uso di un utente finale, poiché incompleto nelle funzionalità sviluppate;
  - `rc`: **r**elease **c**andidate, versione candidata al rilascio
    - software che contiene tutti o una parte dei requisiti obbligatori richiesti;
    - passa tutte le tipologie di test;
    - si presta all'uso di un utente finale;
  - `stable`: versione **stabile**, pronta al rilascio pubblico
    - software completo che implementa tutti i requisiti obbligatori;
    - passa tutte le tipologie di test ed è validato;
    - può essere collaudato con il proponente e/o pubblicato.

Una versione del software subisce un incremento di stato in base alla sua completezza in termini di funzionalità e requisiti soddisfatti. In particolare:

- una versione può ricevere lo stato di `rc` solo se il primo numero (δ) o il secondo numero (ϵ) è diverso da 0;
- una versione può ricevere lo stato di `stable` solo se il primo numero (δ) è diverso da 0.

Esempi codice di versionamento

- `v0.3.5-dev` : versione non completa, non usabile e in parte funzionante;
- `v0.7.0-rc` : versione non completa, usabile, funzionante e con alcuni requisiti implementati;
- `v1.0.0-rc` : versione completa, usabile, funzionante, verificata e validata;
- `v1.0.0-stable` : versione completa, usabile, funzionante, verificata e validata, pronta al rilascio o al collaudo.

Rilascio

Le release del software vengono eseguite internamente nel REPOSITORY$_G$ in base alle funzionalità sviluppate. Inoltre, i rilasci interni saranno normati come segue:

- le versioni `stable` e/o MAJOR RELEASE$_G$ devono essere approvate dall'AMMINISTRATORE$_G$ e dal RESPONSABILE$_G$;
- le versioni `rc` e/o MINOR RELEASE$_G$ devono essere approvate, previa notifica di conferma di tutti i TEST$_G$, dall'AMMINISTRATORE$_G$ e dai VERIFICATORI$_G$;
- le versioni `dev` e/o PATCH$_G$ non richiedono approvazione e possono essere rilasciate autonomamente dal programmatore.

Integrazione con il versionamento di prodotto

Il software, che è formato da diverse parti con le proprie evoluzioni, viene considerato come componente del prodotto. Pertanto, nel codice di versionamento viene aggiunto in coda l'identificativo della baseline di prodotto su cui si basa. Questa prassi non si applica alla nomenclatura dei nomi dei file che riportano la versione.

3.2.4.4 Continuous Integration

L'attività di *Continuous Integration* viene praticata per eseguire degli incrementi a livello di funzionalità delle componenti del prodotto. In particolare, permette di controllare l'aggiunta e la rimozione di parti di codice con i test che vengono appositamente creati.

La *Continuous Integration* permette quindi di lavorare in modo sufficientemente sicuro, così da avere a disposizione un prodotto sempre funzionante che viene costruito a piccoli pezzi, i cui errori possono essere facilmente identificati e revisionati da parte dei programmatori e dei verificatori.

Github Actions

Nell'ambito del progetto, si fa uso delle GITHUB ACTIONS$_G$ per eseguire l'attività di integrazione continua. Una *action* è un insieme di uno o più operazioni (*job*) che devono essere eseguiti su una istanza virtuale di un sistema operativo. È importante fare riferimento a quanto segue:

- una singola operazione, per considerarsi corretta, deve avere un codice di uscita (*exit code*) positivo (0);
- il fallimento di una GITHUB ACTIONS$_G$ può avvenire nel momento in cui il processo venga manualmente annullato, oppure una operazione ha avuto un codice di uscita negativo (1);
- una operazione può essere eseguita sulla istanza corrente del sistema operativo selezionato dalla configurazione corrente;
- gran parte delle operazioni vengono eseguite su DOCKER$_G$ in contenitori indipendenti e su nuove istanze appositamente generate;
- una *action* può essere avviata sul repository ospitante.

La struttura sequenziale delle operazioni è la seguente:

- configurazione delle operazioni da eseguire;
- esecuzione delle operazioni con riscontro dei codici di uscita;
- esito finale e notifica sui canali telematici agli attori interessati.

Documentazione

Il repository è stata configurata per garantire una buona accessibilità e correttezza nello sviluppo dei documenti di progetto da parte dei membri del team. A tal proposito, è stato messo in atto un processo di integrazione continua in ambiente di sviluppo nel momento in cui si vuole eseguire un incremento di versione di un documento. Seguendo la configurazione del workflow adottata per il progetto, a partire da una modifica di un documento che viene eseguito su un branch `feature/`, il documento deve essere sempre compilabile in LaTeX. Viene quindi eseguito un controllo preventivo prima di permettere l'integrazione finale delle modifiche tali da essere convalidate.

1. ad ogni *pull request* da un branch `feature` al `develop`, viene eseguita una BUILD$_G$ di controllo di tutti i documenti modificati;

2. se è tutto corretto, la BUILD$_G$ crea un ARTEFATTO$_G$ con tutti i documenti PDF, che viene salvato nel repository e reso disponibile per essere visionato da remoto:
   - artifacts.redroundrobin.site
3. tramite gli appositi canali di comunicazione, vengono notificati tutti i membri del team delle nuove modifiche ai documenti.

Nel caso di **errori** nella compilazione dei file, viene inviato un avviso all'ultima persona che ha eseguito il COMMIT$_G$ e vengono mantenuti gli ultimi file correttamente compilati.

Componenti software

I singoli componenti fanno uso delle GITHUB ACTIONS$_G$ per l'attività di integrazione continua. In particolare, in base alla tipologia di componente, vengono create delle operazioni di controllo basate sui TEST$_G$ per verificare i requisiti del progetto.

3.2.4.5 Notification & monitoring

L'attività di *notification & monitoring* permette di notificare i membri del team al fine di avere sempre sotto controllo le modifiche effettuate in un componente del prodotto. Questo, ad esempio, permette di venire a conoscenza dei fallimenti o successi per l'attività di *Continuous Integration*. In questo caso, notifiche più specifiche possono essere inviate solamente ai membri interessati che sono stati coinvolti nel processo di modifica della componente.
Generalmente, si usano i seguenti canali telematici per le notifiche:

- pagina delle GitHub Actions del repository;
- email automatiche;
- canali Slack.

Il **monitoraggio** viene eseguito principalmente per mezzo di notifiche. Tuttavia, poiché il coordinamento risulta essere di essenziale importanza, si è deciso di integrare uno strumento che traccia in tempo reale gli ultimi avvenimenti all'interno di un repository da parte dei singoli membri del gruppo. Questo strumento, denominato *Workflow Tracker*, permette di individuare l'ultimo branch in cui un utente ha lavorato, così da ridurre preventivamente i conflitti delle modifiche e sapere le ultime attività all'interno del repository.
Lo strumento è pubblicamente visibile e reperibile al seguente indirizzo per tutti i membri del gruppo:

- workflow.redroundrobin.site

Il *workflow tracker* aggiorna automaticamente ogni 20 secondi gli ultimi movimenti all'interno di una repository e identifica tutti gli autori che hanno eseguito un'azione di *push* all'interno del repository.

3.2.5 Metriche

La gestione della configurazione non fa uso di particolari metriche per misurare la qualità.

3.2.6 Strumenti

3.2.6.1 Repository

Il repository è un luogo remoto in cui vengono mantenuti, salvati e versionati tutti i file che riguardano il progetto per tutto il ciclo di vita del prodotto.
Questo risiede in modo condiviso all'interno di GITHUB$_G$ ed è accessibile solamente ai membri del team.

Versionamento con GitHub

Il repository fa uso di un *Version Control System* (VCS) di tipo distribuito sotto il motore *Git*, che permette la condivisione dal locale al remoto del proprio spazio di lavoro su un luogo comune.
Attraverso l'utilizzo di un web browser, è possibile collegarsi a *GitHub* e controllare i file contenuti in un repository.

Configurazione del workflow

Usando *Git*, è possibile clonare e scaricare da remoto tutto il contenuto del repository per averne una copia in locale su cui poter lavorare, visionando la cronologia dei file modificati ad ogni COMMIT$_G$ da parte di un membro del gruppo.
Git, inoltre, include la possibilità di creare dei BRANCH$_G$ (locali e remoti) in cui poter sviluppare in maniera indipendente una funzionalità, che potrà essere integrata successivamente, senza bisogno di stare al passo con gli aggiornamenti del repository.

Pertanto, si è deciso di stabilire il seguente canone di WORKFLOW$_G$ per quanto concerne la documentazione:

- `master` branch: branch principale su cui vengono fatti i rilasci ad ogni MILESTONE$_G$;
- `develop` branch: branch di sviluppo su cui viene fatta integrazione di nuove funzionalità concluse;
- `feature/xxxx` branch: branch indipendente usato da uno o più membri del gruppo per sviluppare una sezione o fare una revisione di un documento;
  - **xxxx:** si riferisce al numero della ISSUE$_G$ o al titolo della funzionalità da integrare.
  - il *feature branch* riferisce generalmente una singola ISSUE$_G$ ed è rimosso alla sua chiusura.

Ad ogni milestone viene associata una release interna al repository, così da tenere traccia di una BASELINE$_G$.

Cartelle di configurazione

Il repository si compone di diverse tipologie di cartelle. Oltre alle cartelle che riguardano la documentazione e il software, è presente una cartella che viene usata ai fini della configurazione di GITHUB$_G$.

- `.github/`: cartella per il repository di GITHUB$_G$ con i file di configurazione per le GITHUB ACTIONS$_G$ e l'ISSUE TRACKING SYSTEM$_G$.

Formati dei file per le configurazione

In generale, vengono utilizzati alcuni file con formati speciali per la configurazione del repository. Questi file vanno modificati solamente dall'AMMINISTRATORE$_G$, o su richiesta, anche da parte di altri membri del gruppo, in base alle necessità.

- **.gitignore** contiene tutte le regole per evitare di caricare nel repository dei formati non autorizzati (es: file eseguibili);
- **.yml** contiene la configurazione di una GITHUB ACTION$_G$ per dirigere il WORKFLOW$_G$;
- **file senza formati** contengono configurazioni aggiuntive per GITHUB$_G$ (es: template delle ISSUE$_G$);

Formati dei file per i documenti

Per la documentazione si usano principalmente i seguenti tipi di file:

- **file .tex:** contiene il codice sorgente LaTeX del documento;
- **file .pdf:** è il documento compilato;
- **file .png:** identifica una immagine;
- **file .md:** identifica un file scritto in MARKDOWN$_G$, generalmente usato per gli appunti.

3.2.6.2 Project board

Al fine di ottenere un coordinamento il più possibile trasparente dei compiti di un singolo membro del gruppo, si è deciso di introdurre, attraverso la suite di GITHUB$_G$, una **project board** generale che permette di visionare l'andamento dei task, dalla loro creazione fino al loro completamento.
La project board può essere acceduta e visionata solamente dai membri del team al seguente indirizzo:

http://project.redroundrobin.site

Issue tracking system

GitHub integra un *Issue Tracking System* (ITS) che permette di tracciare lo sviluppo tramite ticketing, assegnando a ciascun membro del team una o più ISSUE$_G$ in base alle necessità.
L'ITS viene attivato su ogni repository di progetto e ciascuno viene poi collegato nella project board generale.
Una ISSUE$_G$ descrive un problema che deve essere risolto, sia riguardante l'introduzione di una nuova funzionalità, sia per la risoluzione di un bug. Inoltre, una qualunque ISSUE$_G$ può essere:

- assegnata a uno o più membri del team;
- aperta o chiusa;
- assegnata a una milestone;
- assegnata a una project board;
- discussa a mo' di *forum* da tutti i membri del gruppo.

3.2.6.3 Git submodules

Analizzando il quantitativo di componenti software, sono stati individuati dei possibili problemi che potrebbero sorgere qualora si decidesse di mantenere tutte le componenti del prodotto in un solo luogo.
Un grande quantitativo di cartelle può causare confusione e rallentamento nello sviluppo, specialmente nel momento in cui si rende necessario **collaborare** tramite i branch.
Decidendo di lavorare con un workflow basato sui *feature branch*, il fatto di lavorare su più componenti può portare a un **livello di astrazione maggiore**, che può comportare problemi di coordinamento e conflitti nelle componenti realizzate.

Introduzione ai sotto-moduli

Al fine di porre rimedio a questa eventuale problematica, si è deciso di introdurre una funzionalità di GIT$_G$ per poter separare lo sviluppo delle componenti software in più **sotto-moduli**.
Ciascun sotto-modulo viene identificato come un repository indipendente, dove viene sviluppata una componente software seguendo la stessa configurazione di *workflow*.
Tutti i sotto-moduli possono essere singolarmente recuperati e versionati in un repository principale, in cui si va a comporre l'intero prodotto seguendo l'andamento di rilascio delle relative *baseline* di prodotto.

- un sotto-modulo si identifica come repository di GITHUB$_G$;
- un sotto-modulo può essere aggiornato nel repository principale seguendo gli ultimi commit nel branch principale o recuperando la versione di rilascio;
- nel repository principale, solo l'amministratore aggiorna manualmente le versioni dei componenti (o sotto-moduli) in base ai rilasci di prodotto.

Repository principale

Il repository principale si compone di tutti i sotto-moduli che vengono versionati in base all'ultimo commit nel sotto-modulo o in base alla versione rilasciata del sotto-modulo.
La struttura di dipendenza dei repository è la seguente:

- `swe-thirema`: repository principale con il prodotto intero
  - `swe-docs`: sotto-modulo per la documentazione;
  - `swe-webapp`: sotto-modulo per il sito web di progetto;
  - `swe-api`: sotto-modulo per le API$_G$;
  - `swe-telegram`: sotto-modulo per il BOT TELEGRAM$_G$;
  - `swe-kafka-db`: sotto-modulo per i database e APACHE KAFKA$_G$;
  - `swe-gateway`: sotto-modulo per il GATEWAY$_G$ e i dispositivi.

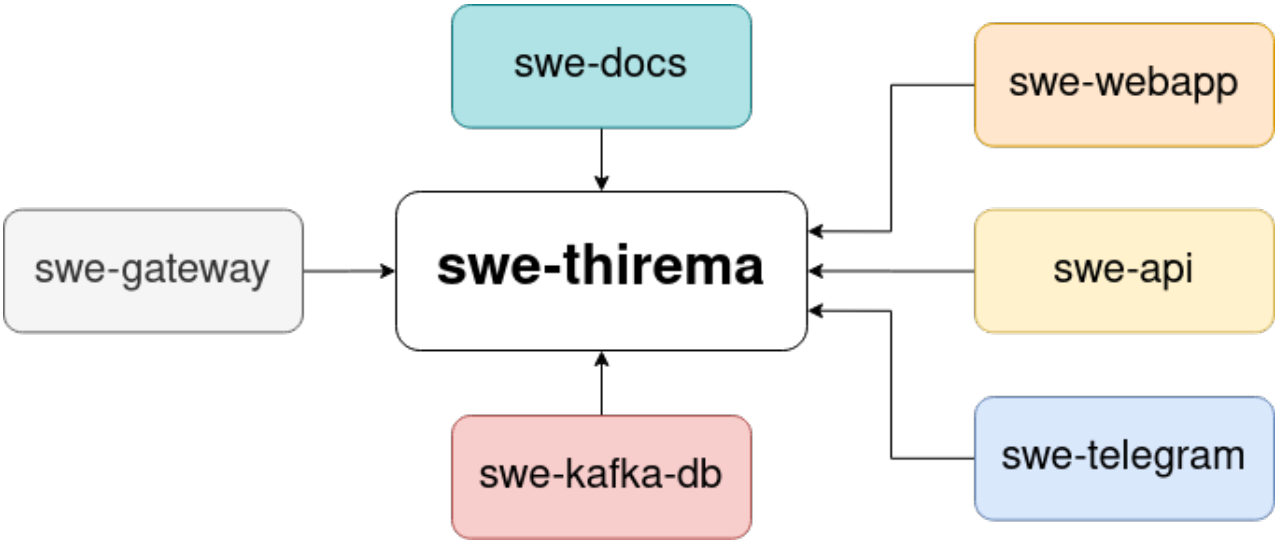

Figura 1: Grafico riassuntivo dei Git submodules utilizzati nel progetto

Il repository principale può essere acceduta dal seguente indirizzo:

thirema.redroundrobin.site

Sotto-modulo swe-docs

- **descrizione:** contiene tutta la documentazione di progetto;
- **nome completo:** `RedRoundRobin/swe-docs`;
- **URL:** docs.redroundrobin.site;
- **linguaggi di sviluppo:** LaTeX;
- **struttura delle cartelle:**
  - `interni/`: contiene tutta la documentazione interna;
  - `esterni/`: contiene tutta la documentazione esterna;
  - `template/`: contiene tutti i template delle varie tipologie di documento;
  - `notes/`: contiene tutte le note aggiuntive (*non*-LaTeX) sui seminari, le guide e l'organizzazione.

Sotto-modulo swe-webapp

- **descrizione:** contiene il sito web richiesto dal progetto;
- **nome completo:** `RedRoundRobin/swe-webapp`;
- **URL:** webapp.redroundrobin.site;
- **linguaggi di sviluppo:** PHP$_G$, JAVASCRIPT$_G$.

Sotto-modulo swe-api

- **descrizione:** contiene le API$_G$ per la comunicazione tra APACHE KAFKA$_G$ e le applicazioni esterne;
- **nome completo:** `RedRoundRobin/swe-api`;
- **URL:** api.redroundrobin.site;
- **linguaggi di sviluppo:** JAVA$_G$;
- **documentazione online API (branch *develop*):** api.docs.redroundrobin.site.

Sotto-modulo swe-telegram

- **descrizione:** contiene il BOT TELEGRAM$_G$;
- **nome completo:** `RedRoundRobin/swe-telegram`;
- **URL:** telegram.redroundrobin.site;
- **linguaggi di sviluppo:** JAVASCRIPT$_G$.

Sotto-modulo swe-kafka-db

- **descrizione:** contiene il componente per la configurazione di KAFKA$_G$ e dei database;
- **nome completo:** `RedRoundRobin/swe-kafka-db`;
- **URL:** kafka-db.redroundrobin.site;
- **linguaggi di sviluppo:** SQL$_G$.

Sotto-modulo swe-gateway

- **descrizione:** contiene l'implementazione di un GATEWAY$_G$ con l'integrazione di un dispositivo;
- **nome completo:** `RedRoundRobin/swe-gateway`;
- **URL:** gateway.redroundrobin.site;
- **linguaggi di sviluppo:** JAVA$_G$.

3.2.6.4 Tecnologie di supporto e software

Le tecnologie e i software coinvolti per la configurazione del workflow si pongono i seguenti obiettivi:

- uniformare il *modus operandi* del gruppo, consentendo di collaborare su programmi comuni e multi-piattaforma;
- automatizzare il lavoro per delegare gli impieghi che possono essere svolti tramite macchina, velocizzando lo sviluppo.

Programmi per la gestione del repository

Il repository viene gestita principalmente in un server remoto, accessibile tramite il portale di Github.com da tutti i membri del team.
Il processo di sviluppo, tuttavia, avviene in locale, eseguendo una clonazione del repository con tutti i branch attivi tramite uno dei seguenti programmi:

- **Github Desktop:** applicazione ufficiale di Github, molto semplice e leggera, disponibile per Linux, Mac OS e Windows;
- **GitKraken:** applicazione compatibile e più avanzata di Github desktop, disponibile per tutte le piattaforme;
- **Git CLI:** applicazione da terminale, disponibile per tutte le piattaforme con tutti i comandi utili per Git.

Per ciascun programma viene richiesta autenticazione, che può avvenire tramite semplice login (username e password) o tramite generazione di chiavi SSH, successivamente importate tramite una procedura guidata su GITHUB$_G$.

Automazione del workflow

Il repository, che fa uso di GITHUB$_G$, integra nativamente uno strumento denominato GITHUB ACTIONS$_G$ che permette di configurare le seguenti attività:

- **Continuous Integration:** integrazione continua del software con controllo sui TEST$_G$;
- **Continuous Delivery:** consegna continua del software sfruttando gli incrementi minori per eseguire test più spesso;
- **Continuous Deployment:** distribuzione continua e messa in funzione del software;
- **notification & monitoring:** invio di notifiche ai membri del team e monitoraggio delle attività di DEVOPS$_G$.

Nel nostro caso, le GITHUB ACTIONS$_G$ permettono di fare in un solo file di configurazione tutti i passaggi in base al WORKFLOW$_G$ che ci interessa integrare.
Delle operazioni descritte precedentemente, si fa uso solamente di:

- Countinuous Integration;
- notification & monitoring.

I rimanenti processi verranno eventualmente impiegati per la manutenzione del prodotto.

3.2.6.5 Automazione delle build e delle dipendenze

Per ciascun componente software, si è deciso di integrare degli strumenti di building automation col fine di semplificare la gestione delle dipendenze e l'avvio dei processi di automazione attraverso le GITHUB ACTIONS$_G$.
In particolare, si è deciso di utilizzare:

- `maven`: utilizzato in ambiente Java, molto conosciuto e facilmente integrabile;
- `npm`: *node package manager*, usato in ambiente Javascript;
- `composer`: usato in ambiente PHP, integrato con il framework *Laravel*.

Tutti questi strumenti sono stati configurati, rispettivamente, all'interno di ciascuna componente software che ne fa uso, con delle regole per l'analisi statica, versioni dei pacchetti da utilizzare e suite test di default da eseguire.

	maven	**npm**	**composer**
swe-docs			
swe-webapp		✓	✓
swe-api	✓		
swe-telegram		✓	
swe-kafka-db	✓		
swe-gateway	✓		

Tabella 2: Tabella riassuntiva dei componenti software che fanno uso degli strumenti di building automation e gestione delle dipendenze.

Maven

Maven è uno strumento sviluppato da *Apache* e utilizzato per la gestione del progetto, esso permette di configurare tutte le dipendenze e tutti i plugin utili allo sviluppo di una applicazione *Java*.
Integrato con *IntelliJ Idea*, permette di configurare automaticamente il repository di progetto, scaricando e cancellando autonomamente le dipendenze necessarie, in base a quanto riportato nel `pom.xml`.
Questo file si trova nella cartella principale del *git submodule* in cui viene sviluppata l'applicazione *Java* e può essere modificato in base alle necessità. Di seguito si mostra un esempio di `pom.xml`:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <project xmlns="http://maven.apache.org/POM/4.0.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>groupId</groupId>
        <artifactId>swe-gateway</artifactId>
        <version>1.0-SNAPSHOT</version>

        <dependencies>
            <dependency>
                <groupId>org.apache.kafka</groupId>
```

```
                <artifactId>kafka_2.13</artifactId>
                <version>2.4.0</version>
            </dependency>

            <dependency>
                <groupId>org.jetbrains</groupId>
                <artifactId>annotations-java5</artifactId>
                <version>RELEASE</version>
                <scope>compile</scope>
            </dependency>

            <!-- ... -->
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.0</version>
                    <configuration>
                        <release>11</release>
                    </configuration>
                </plugin>
            </plugins>
        </build>
    </project>
```

Snippet 4: Esempio di implementazione di un file pom.xml

npm - Node Package Manager

Node Package Manager viene utilizzato in ambiente Javascript per la gestione delle dipendenze.
Il suo punto di forza risulta essere la gestione delle componenti, le quali possono essere salvate all'interno di un file *json*, denominato *package.json*.
Questo file viene utilizzato per salvare la configurazione di tutti i moduli che si vogliono integrare nel proprio progetto e che devono essere compilati a corredo del prodotto.
Per ciascun modulo vengono risolte poi le dipendenze attraverso un nuovo file denominato *package-lock.json*, che viene versionato assieme al *package.json*. In aggiunta:

- le installazioni e le rimozioni di moduli possono essere fatte tramite **npm-cli**, il programma ufficiale di npm a riga di comando;
- ogni volta che si esegue un aggiornamento delle dipendenze, vengono mostrate le ultime vulnerabilità scoperte con le relative correzioni;
- la compatibilità è assicurata su qualsiasi sistema operativo e soprattutto con gli IDE usati nel progetto;

- la cartella `node_modules/` che viene generata alla prima configurazione di *npm* **non** deve essere versionata.

Composer

Composer viene utilizzato come gestore delle dipendenze in ambiente PHP.
Associato al framework LARAVEL$_G$, permette di integrare, in modo affine a *npm*, le stesse funzionalità. Infatti, viene utilizzata una interfaccia a riga di comando, da cui eseguire tutte le operazioni di installazione, aggiornamento e rimozione dei moduli usati; si ritrova, anche in questo caso, le notifiche di vulnerabilità delle dipendenze installate.
Composer utilizza un file *composer.json* che contiene tutti i componenti che vengono utilizzati a corredo dell'applicazione. In aggiunta:

- le installazioni e le rimozioni di moduli possono essere fatte a riga di comando tramite il comando `composer`;
- la compatibilità è assicurata su qualsiasi sistema operativo e soprattutto con gli IDE usati nel progetto;
- la cartella `vendor/` che viene generata alla prima configurazione di *composer* **non** deve essere versionata.

3.2.6.6 Strumenti video

Al fine di eseguire registrazioni di video dimostrativi per il proponente e per il manuale utente, sono stati utilizzati degli strumenti per la registrazione dello schermo e per il montaggio finale. In particolare, si è fatto uso di strumenti gratuiti e con molteplice compatibilità a livello di sistema operativo.

OBS

Il software OBS è stato utilizzato per eseguire le registrazione dei contenuti del video.
Il software è molto semplice e intuitivo da usare dal momento che è possibile avviare la registrazione di una sorgente (monitor) o di un'unica porzione di schermo.
Il video registrato può essere poi salvato con formati differenti, tipicamente `.mp4`, ed esportato su una cartella di destinazione.

DaVinci Resolve

Il software DaVinci Resolve è uno strumento di editing e montaggio video con diversi livelli di complessità.
Nell'ambito del progetto, si rende sufficiente utilizzare la modalità *Cut* che permette l'aggregazione delle clip audio e video in modo del tutto intuitivo.
Le operazioni basilari che si possono svolgere in questa modalità riguardano:

- l'aggregazione di più clip video in modo sequenziale;
- l'aggiunta di più tracce audio, anche sovrapposte;

- il ritaglio delle clip video e audio;
- l'inserimento di transizioni tra più clip video.

Il video, una volta completato, viene poi esportato in formato `.mp4` o in un formato differente in base alle esigenze.

YouTube

YouTube è una piattaforma online di pubblicazione e streaming video che si integra nativamente con un account Google. L'inserimento dei video è gratuito e il suo utilizzo è molto intuitivo.
Attraverso l'uso dell'account Google di progetto, si è deciso di aprire un canale YouTube in forma privata col fine di pubblicare i video di presentazione per il proponente e per il manuale utente.
I video pubblicati all'interno del canale di progetto sono solamente in forma privata e accessibili unicamente tramite link.

3.3 Gestione dei cambiamenti

3.3.1 Scopo

L'obiettivo del processo di gestione dei cambiamenti è fornire un mezzo tempestivo, efficace e documentato per assicurarsi che tutti i problemi vengano analizzati, documentati, risolti e, in un'ottica di miglioramento continuo, evitati.

3.3.2 Aspettative

Avviando questo processo si intende ottenere:

- una qualità della documentazione e del codice più elevata;
- un modo efficiente ed efficace per trovare e correggere errori evitando che si propaghino;
- un tracciamento continuo degli errori più comuni e delle loro fonti, in modo tale da risolverli all'origine.

3.3.3 Descrizione

Il processo di gestione dei cambiamenti definisce una procedura da seguire per analizzare e rimuovere dei problemi, qualsiasi sia la loro origine, scoperti durante l'esecuzione del processo di sviluppo, di manutenzione o di altri processi.

3.3.4 Attività

3.3.4.1 Implementazione del processo

Il processo di gestione dei cambiamenti dovrà essere eseguito ogni volta che sarà necessario gestire un problema e dovrà soddisfare i seguenti requisiti:

1. il processo dovrà essere un ciclo chiuso, assicurandosi che:
   - tutti i problemi rilevati dovranno essere prontamente riportati ed immessi nel processo di gestione dei cambiamenti;
   - dovranno essere avvisate le parti interessate;
   - le cause dovranno essere identificate, analizzate e nel limite del possibile rimosse;
2. il processo utilizzerà uno schema per categorizzare e dare la giusta priorità ai problemi scovati;

Identificativo	Processo	Osservazione	Soluzione

l'identificativo è composto da:

CMB-[tipologia][priorità]-[numero]

dove **CMB** identifica un cambiamento che è stato attuato;

la **tipologia** potrà avere i seguenti valori:

- **O:** per i problemi ortografici;
- **C:** per problemi legati al contenuto;

la **priorità** avrà invece i seguenti valori:

- **A:** alta, richiede una tempestiva risoluzione;
- **M:** media, per problemi la cui risoluzione potrebbe aggravarsi nel tempo;
- **B:** bassa, se il problema ha una possibilità molto ridotta di aggravarsi o provocare altri problemi;

il **numero** alla fine sarà un numero progressivo che parte da 1;

3. verranno effettuate delle analisi per verificare la presenza di tendenze nei problemi riportati;
4. il procedimento di risoluzione dei problemi verrà valutato: si valuterà che i problemi siano stati effettivamente risolti, che le eventuali tendenze siano state annullate ed infine che non siano stati introdotti altri errori.

3.3.4.2 Risoluzione del problema

Quando uno o più problemi saranno rilevati, nel prodotto software o in un'attività, dovrà essere preparato un report aprendo una issue nell'ITS (Issue tracking System) utilizzato per ogni problema individuato. Lo stesso report dovrà essere parte integrante del procedimento che è stato descritto nell'attività di istanziazione del processo.

3.3.5 Metriche

La gestione dei cambiamenti non fa uso di particolari metriche per misurare la qualità.

3.3.6 Strumenti

3.3.6.1 GitHub

Per creare report si è deciso di utilizzare l'ITS fornito da GitHub$_G$, in quanto fornisce anche un sistema di versionamento ed è conosciuto da tutti i membri del gruppo. Questo ITS fornisce inoltre la possibilità di collegare i commit del cambiamento con la issue legata al report, in modo tale da poter tracciare tutto il percorso dalla modifica alla risoluzione del problema.

3.4 Garanzia della qualità

3.4.1 Scopo

Si occupa di stabilire una metrica precisa per tutti i servizi nell'ambito della verifica e della validazione, mantenendo un dato livello di qualità che rimanga uniforme e misurabile durante tutto il ciclo di vita del software.

3.4.2 Aspettative

Il sistema di qualità deve fornire delle metriche di giudizio uniformi volte a quantificare con chiarezza la correttezza dei documenti e del software. Ciò va unito anche all'affidabilità nello svolgimento dei processi di verifica, che vanno monitorati e guidati nell'intera procedura, senza lasciare a interpretazioni. Pertanto, ci si aspetta:

- un prodotto software di qualità;
- una documentazione completa e facilmente comprensibile per tutti;
- dei processi che seguano dei punti ben specificati per l'analisi di qualità;
- una comunicazione chiara e semplice delle problematiche relative alla qualità tra i membri del team;
- una registrazione dei risultati ottenuti.

3.4.3 Descrizione

La garanzia della qualità si compone di diversi controlli che devono essere effettuati per:

- il software;
- la documentazione;
- tutti i processi che portano alla realizzazione della documentazione e del software.

Per ogni processo mirato alla qualità si definiscono delle metriche che vengono riportate in ciascuna sezione del presente documento. Le registrazioni dei risultati ottenuti dall'analisi della qualità sono salvate con degli appositi report.

3.4.3.1 Obiettivi di qualità di prodotto

La qualità del prodotto viene garantita attraverso l'attuazione dei processi di verifica e validazione basati su fondamenti normativi. In particolare, definiamo quanto segue:

- **verifica:** processo di controllo che garantisce qualità dei processi di fornitura del prodotto;

- **validazione:** processo di controllo del prodotto volto a confermare le aspettative, i requisiti e le funzionalità concordate.

L'insieme di questi processi deve portare a un miglioramento continuo del prodotto, che viene sottoposto agli standard di qualità riportati nel *way of working*.

3.4.3.2 Obiettivi qualità di processo

La qualità di processo deve essere perseguita nel corso del ciclo di vita del software attraverso i principi di efficacia ed efficienza mirati al prodotto. Nello specifico definiamo quanto segue:

- **efficacia:** si richiede un prodotto che soddisfi le richieste del proponente;
- **efficienza:** i processi devono convergere con costi ridotti in termini di risorse a pari qualità di prodotto.

Ciascun processo va migliorato durante la sua esecuzione facendo uso di monitoraggi mirati che permettano di acquisire, attraverso l'esperienza, una risposta critica alla qualità stessa del processo.

3.4.4 Attività

3.4.4.1 Classificazione dei processi

Nell'ambito di qualità, i processi vengono tracciati con il seguente identificativo:

$$\text{QP} - [\lambda]$$

Dove:

- QP: indica letteralmente *Quality Process*;
- λ: numero intero che indica il processo e parte da 1.

3.4.4.2 Classificazione delle caratteristiche di prodotto

Nell'ambito di qualità, le caratteristiche del prodotto vengono tracciate con il seguente identificativo:

$$\text{QC} - [\lambda]$$

Dove:

- QC: indica letteralmente *Quality Characteristic*;
- λ: numero intero che indica la caratteristica di prodotto e parte da 1.

3.4.4.3 Classificazione delle metriche

Le metriche sono i criteri che vengono utilizzati per misurare nei processi e nel prodotto i gradi di qualità raggiunti. A ciascuna metrica si associa il seguente identificatore:

$$\text{QM} - [\delta] - [\lambda]$$

Dove:

- QM: indica letteralmente *Quality Metric*;
- δ: riguarda la tipologia della metrica e può assumere i valori riportati di seguito:
  - **PROC**: indica che la metrica si associa per un processo;
  - **PROD**: indica che la metrica si associa per il prodotto;
  - **TEST**: indica che la metrica si associa per i test;
- λ: numero intero che indica la metrica e parte da 1.

3.4.5 Metriche

Per ciascuna attività, sia che riguardi la documentazione, il software o il monitoraggio di processo, si riporta nella relativa sezione una classificazione delle metriche di qualità.

3.4.5.1 QP-5 Garanzia della qualità

Scopo

Durante le fasi di progetto, specialmente quelle che riguardano lo sviluppo, si rende necessario monitorare il quantitativo delle metriche soddisfatte per comprendere l'attenzione alla qualità da parte del gruppo.

Introduzione alle metriche

Per la garanzia della qualità si farà uso delle seguenti metriche:

- QM-PROC-11 Percentuale delle metriche soddisfatte (PMS).

QM-PROC-11 Percentuale delle metriche soddisfatte (PMS)

- **Descrizione:** la metrica permette di conoscere la percentuale delle metriche soddisfatte fino a quel momento, sulla base delle metriche disponibili;
- **Unità di misura:** la metrica viene espressa in percentuale.

- **Formula:** la formula della metrica è la seguente:

$$\text{PMS} = \frac{\text{Numero di metriche soddisfatte}}{\text{Numero totale di operandi}} \times 100$$

  dove per *metriche soddisfatte* si intendono tutte le metriche la cui soglia di accettabilità è stata superata;

- **Risultato:**
  - se il risultato è pari a 0%, nessuna metrica è stata superata.
  - se il risultato è pari al 100%, tutte le metriche sono state superate.
  - se il risultato è maggiore di 0%, ma minore di 100%, non tutte le metriche hanno raggiunto la soglia di accettabilità.

3.4.6 Strumenti

Non sono stati identificati degli strumenti particolari per la garanzia di qualità.

3.5 Verifica

3.5.1 Scopo

Il processo di verifica ha lo scopo di stabilire se il prodotto è realizzato nel modo corretto secondo delle regole stabilite.

3.5.2 Aspettative

Lo svolgimento del processo di verifica sarà garantito seguendo determinati punti:

- definizione di criteri di accettazione;
- prescrizione delle attività di verifica con relativa documentazione;
- test di verifica;
- correzione di eventuali DIFETTI$_G$.

3.5.3 Descrizione

La verifica consiste nel cercare e risolvere possibili *difetti* all'interno della documentazione e del codice prodotto. Il completamento del processo di verifica rende possibile l'esecuzione del processo di validazione.

3.5.4 Attività

3.5.4.1 Analisi statica e dinamica

Analisi statica

L'analisi statica viene effettuata sul prodotto senza necessità di eseguirlo e serve per verificare che non ci siano errori o *difetti*. I due tipi di analisi statica sono:

- **walkthrough:** consiste nell'analizzare i vari documenti e file in tutto il loro contenuto per trovare eventuali *difetti*. Il verificatore controlla se sono presenti *difetti* e, in caso ne trovi, la correzione verrà effettuata dagli sviluppatori;
- **inspection:** in questa tecnica è noto a priori dove debbano essere ricercati i possibili *difetti* del prodotto, quindi non si analizzano i documenti e file per intero, ma solo le parti in cui di solito sono presenti. Il verificatore compone una lista di controllo (checklist) inserendo i punti in cui si possono rilevare possibili *difetti*, controlla in quei punti della lista e, se trova delle incorrettezze, la correzione viene poi effettuata dagli sviluppatori.
  Di seguito alcuni possibili punti in cui trovare *difetti* all'interno della documentazione:
  - elenchi puntati;

- formato date;
- parole/frasi in grassetto/corsivo;
- uso di riferimenti appropriati al glossario/documento.

Analisi dinamica

L'analisi dinamica è una tecnica per cui è necessaria l'esecuzione dell'oggetto di verifica, e consiste nell'attività di test.

3.5.4.2 Test

I test fanno parte dell'attività di analisi dinamica e servono per individuare possibili errori di funzionamento del codice. Per effettuare i test, essi devono essere automatizzati, tramite strumenti appositi, e ripetibili, ovvero devono essere definiti:

- l'ambiente di sviluppo e lo stato iniziale;
- le istruzioni eseguite;
- i dati di input e i dati di output attesi.

Per elencare le specifiche dei test si è scelta una rappresentazione tabellare contenente il codice del componente da testare, la descrizione dei test, il suo stato di avanzamento e il risultato del test stesso secondo la seguente nomenclatura:
Stato:

- **I:** il test è stato implementato;
- **NI:** il test non è ancora stato implementato.

Risultato:

- **S:** il test ha successo;
- **F:** il test ha fallito.

Tuttavia si è scelto di omettere il risultato dei test in quanto non significativo allo stato attuale del prodotto.

Test di sistema

Dopo aver eseguito i test su tutte le unità e sulla loro integrazione, si testa il sistema nella sua interezza. Viene testato se le interazioni tra le varie componenti del sistema ritornano il risultato atteso o meno in concordanza con ciò che è stato definito nel processo di analisi dei requisiti. Per classificare questa tipologia di test si utilizzerà un codice utilizzando il seguente modello:

TS-[Identificativo]

Dove:

- **Identificativo:** è un numero progressivo il cui obiettivo sarà di contraddistinguere il singolo requisito da testare; parte da 1.

Test di integrazione

Test eseguiti su componenti del software per verificare se l'insieme di unità si interfaccia come dovrebbe. Questo test è eseguito in modo ricorrente: ogni volta che un insieme di unità esegue correttamente, esso viene integrato con altri insiemi di unità, fino al test completo sul sistema. Per classificare questa tipologia di test si utilizzerà un codice utilizzando il seguente modello:

TI-[Identificativo]

Dove:

- **Identificativo:** è un numero progressivo il cui obiettivo sarà di contraddistinguere la singola integrazione da testare; parte da 1.

Test di unità

Test eseguiti sul funzionamento di unità di software in modo automatico: viene definito l'input e l'output atteso per verificare il corretto funzionamento dell'unità. Per classificare questa tipologia di test si utilizzerà un codice secondo il seguente modello:

TU-[Identificativo]

Dove:

- **Identificativo:** è un numero progressivo il cui obiettivo sarà di contraddistinguere la singola unità da testare; parte da 1.

Test di regressione

Test eseguito ogni volta che un'unità viene modificata allo scopo di trovare *difetti* nelle funzionalità già testate, potendo garantire che le funzionalità preesistenti non abbiano cambiato comportamento. Si rieseguono tutti i test necessari affinché si possa essere certi che la modifica non causi il funzionamento scorretto di altre unità collegate all'unità modificata.

3.5.5 Metriche

3.5.5.1 QP-4 Verifica

Scopo

Durante tutta la fase di sviluppo, si vuole monitorare il processo di verifica (software) mettendo in luce aspetti che riguardano la complessità e la copertura di test a livello di codice. Questo può essere utile sia per il cliente, sia per il gruppo, in modo da comprendere l'avanzamento delle attività di verifica del software fino alla data attuale.

Introduzione alle metriche

Per la verifica del software si farà uso delle seguenti metriche:

- QM-TEST-1 Code coverage (COCO);
- QM-TEST-2 Condition coverage (CONCO);
- QM-TEST-3 Line coverage (LOCO);
- QM-TEST-4 Passed test cases percentage (PTCP);
- QM-TEST-5 Failed test cases percentage (FTCP);
- QM-TEST-6 Bug-fixing percentage (BFP);
- QM-TEST-7 Complessità media dei test di classe (CMTC).

QM-TEST-1 Code coverage (COCO)

- **Descrizione:** la metrica verifica la copertura effettuata attraverso una suite di test, rispetto al codice prodotto del programma;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Fonte:** i dati vengono recuperati attraverso le statistiche di SonarCloud$_G$;
- **Risultato:**
  - se il risultato è pari a 0%, la copertura è nulla;
  - se il risultato è pari al 100%, la copertura è totale;
  - se il risultato è maggiore di 0%, ma minore di 100%, la copertura è parziale.

QM-TEST-2 Condition coverage (CONCO)

- **Descrizione:** la metrica verifica che ogni condizione di tipo booleano realizzata con degli operatori logici, venga considerata sia vera che falsa nei test; questo permette di avere una migliore sensibilità sul controllo di flusso del programma;
- **Unità di misura:** la metrica viene espressa in percentuale;

- **Fonte:** i dati vengono recuperati attraverso le statistiche di SonarCloud$_G$;
- **Formula:** la formula della metrica è la seguente:

$$\text{COCOV} = \frac{\text{Numero di operandi eseguiti dai test}}{\text{Numero totale di operandi}} \times 100$$

- **Risultato:**
  - se il risultato è pari a 0%, la copertura è nulla;
  - se il risultato è pari al 100%, la copertura è totale;
  - se il risultato è maggiore di 0%, ma minore di 100%, la copertura è parziale.

QM-TEST-3 Line coverage (LOCO)

- **Descrizione:** la metrica verifica la percentuale di linee di codice che è stata coperta attraverso i test; questo permette di comprendere il quantitativo di attenzione che è stato dato nella scrittura dei test;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Fonte:** i dati vengono recuperati attraverso le statistiche di SonarCloud$_G$;
- **Risultato:**
  - se il risultato è pari a 0%, la copertura è nulla;
  - se il risultato è pari al 100%, la copertura è totale;
  - se il risultato è maggiore di 0%, ma minore di 100%, la copertura è parziale.

QM-TEST-4 Passed test cases percentage (PTCP)

- **Descrizione:** la metrica PTCP si utilizza per misurare la percentuale di test passati con successo in una specifica fase del progetto fino alla data corrente;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{PTCP} = \frac{\text{Numero di test passati}}{\text{Numero totale di test eseguiti}} \times 100$$

- **Risultato:**
  - se il risultato è pari a 0%, nessun test realizzato per il software è andato a buon fine;
  - se il risultato è pari al 100%, tutti i test realizzati per il software sono andati a buon fine;
  - se il risultato è compreso tra 0% e 100%, non tutti i test realizzati per il software sono andati a buon fine.

QM-TEST-5 Failed test cases percentage (FTCP)

- **Descrizione:** la metrica FTCP viene usata per misurare la percentuale di test falliti in una specifica fase del progetto fino alla data corrente;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{FTCP} = \frac{\text{Numero di test falliti}}{\text{Numero totale di test eseguiti}} \times 100$$

- **Risultato:**
  - se il risultato è pari a 0%, non ci sono test realizzati per il software che sono falliti;
  - se il risultato è pari al 100%, tutti i test realizzati per il software sono falliti;
  - se il risultato è compreso tra 0% e 100%, non tutti i test realizzati per il software sono andati a buon fine.

QM-TEST-6 Bug-fixing percentage (BFP)

- **Descrizione:** la metrica BFP si utilizza per misurare il quantitativo di errori corretti nel codice, rispetto agli errori trovati, fino alla data corrente;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula della metrica è la seguente:

$$\text{BFP} = \frac{\text{Numero di difetti corretti}}{\text{Numero di difetti trovati}} \times 100$$

- **Risultato:**
  - se il risultato è pari a 0%, nessun difetto è stato risolto;
  - se il risultato è pari al 100%, tutti i difetti sono stati risolti;
  - se il risultato è compreso tra 0% e 100%, non tutti i difetti sono stati corretti.

QM-TEST-7 Complessità media dei test di classe (CMTC)

- **Descrizione:** la metrica CMTC viene utilizzata per comprendere il quantitativo medio di test realizzati in relazione al numero di classi presenti nel componente software;
- **Unità di misura:** la metrica è espressa tramite un numero decimale;
- **Formula:** la formula della metrica è la seguente:

$$\text{CMTC} = \frac{\text{numero totale di test}}{\text{numero totali di classi}}$$

- **Risultato:** il risultato della formula ha i seguenti significati:
  - se il risultato è pari a 0, allora nel componente non è stato incluso in alcun test;
  - se il risultato è maggiore di 0 e minore di 1, allora in media ci possono essere delle classi prive di test;
  - se il risultato è maggiore di 1, allora in media ciascuna classe ha almeno un test.

3.5.6 Strumenti

3.5.6.1 Programmi per il controllo ortografico

TeXstudio

È stato utilizzato il correttore ortografico integrato nell'IDE$_G$ TeXstudio, utilizzato per la redazione della documentazione.

Sublime Text

Si fa uso dello *spell checker*, in lingua italiana, per la verifica degli errori ortografici nella documentazione di progetto.

3.5.6.2 Analisi statica del codice

Checkstyle

- **componenti software:** swe-api, swe-kafka-db, swe-gateway;
- **linguaggio:** Java.

Checkstyle è un plugin, che viene integrato dal gestore di dipendenze *Maven*, per eseguire automaticamente i controlli di verifica di regole sintattiche per codice Java.
Permette di essere configurato direttamente da un file XML presente nella cartella di progetto, denominato `checkstyle.xml`, contenente tutte le regole da verificare. In particolare:

- ad ogni build viene eseguita l'analisi statica e vengono segnalati tutti gli errori trovati, tramite degli opportuni warning;
- permette di far rispettare lo standard di codifica adottato;
- il file `checkstyle.xml` deve essere versionato.

Prettier

- **componenti software:** swe-telegram, swe-webapp;
- **linguaggio:** JavaScript, Vue.js.

Prettier è uno strumento per il controllo automatico della formattazione del codice scritto in linguaggio JavaScript, esso permette di eseguire dei controlli sul formato del codice, tramite delle regole configurabili.
Oltre ad effettuare il controllo, mette a disposizione anche una funzionalità di riformattazione automatica del codice, attivabile al momento del salvataggio del file modificato; in questo modo facilità l'aderenza allo stile di codifica da parte dei programmatori.

ESLint

- **componenti software:** swe-telegram, swe-webapp;
- **linguaggio:** JavaScript, Vue.js.

ESLint è uno strumento per il controllo di conformità del codice, usato per il linguaggio JavaScript, che permette di verificare l'aderenza a delle precise regole sintattiche.
Analogamente a *Checkstyle*, permette di eseguire dei controlli simili, attraverso un file di configurazione dove devono essere specificate tutte le regole di codifica che si intendono adottare.
Oltre ad effettuare il controllo, mette a disposizione anche una funzionalità di autocorrezione del codice, in base agli errori rilevati, eseguibile tramite un apposito comando da terminale; in questo modo facilità l'aderenza allo stile di codifica da parte dei programmatori.
Inoltre, attraverso la propria estensione per *Prettier*, è possibile integrare quest'ultimo all'interno di ESLint, in modo da eseguire automaticamente anche i controlli di formattazione del codice e gli eventuali aggiustamenti necessari. In particolare:

- ad ogni commit viene eseguita l'analisi statica e vengono segnalati tutti gli errori trovati, tramite degli opportuni warning;
- permette di far rispettare lo standard di codifica adottato;
- il file `.eslintrc.json` deve essere versionato.

PHP_CodeSniffer

- **componenti software:** swe-webapp;
- **linguaggio:** PHP.

PHP_CodeSniffer è uno strumento per il controllo di conformità del codice, usato per il linguaggio PHP, che permette di verificare l'aderenza a delle precise regole sintattiche, definite da uno standard di codifica.
Analogamente a *Checkstyle*, permette di eseguire dei controlli simili, attraverso un file di configurazione dove devono essere specificate tutte le regole di codifica che si intendono adottare.
Oltre ad effettuare il controllo, mette a disposizione anche una funzionalità di autocorrezione del codice, in base agli errori rilevati, eseguibile tramite un apposito comando da terminale; in questo modo facilità l'aderenza allo stile di codifica da parte dei programmatori.
Tuttavia, per avere una copertura più completa dello standard di codifica definito per PHP, è stato necessario integrare un altro strumento di controllo sintattico che coprisse le regole non verificate da PHP_CodeSniffer. In particolare:

- ad ogni commit viene eseguita l'analisi statica e vengono segnalati tutti gli errori trovati, tramite degli opportuni warning;
- permette di far rispettare lo standard di codifica adottato;
- il file `.phpcs.xml` deve essere versionato.

PHP-CS-Fixer

- **componenti software:** swe-webapp;
- **linguaggio:** PHP.

PHP-CS-Fixer è uno strumento per il controllo di conformità del codice, usato per il linguaggio PHP, che permette di verificare l'aderenza a delle precise regole sintattiche, definite da uno standard di codifica.
Analogamente a *Checkstyle*, permette di eseguire dei controlli simili, attraverso un file di configurazione dove devono essere specificate tutte le regole di codifica che si intendono adottare.
Oltre ad effettuare il controllo, mette a disposizione anche una funzionalità di autocorrezione del codice, in base agli errori rilevati, eseguibile tramite un apposito comando da terminale; in questo modo facilità l'aderenza allo stile di codifica da parte dei programmatori.
Questo strumento è stato affiancato a PHP_CodeSniffer, per avere una copertura più completa dello standard di codifica definito per PHP. In particolare:

- ad ogni commit viene eseguita l'analisi statica e vengono segnalati tutti gli errori trovati, tramite degli opportuni warning;
- permette di far rispettare lo standard di codifica adottato;
- il file `.php_cs.php` deve essere versionato.

SonarCloud

- **componenti software:** swe-api, swe-kafka-db, swe-gateway, swe-telegram, swe-webapp;
- **linguaggio:** tutti.

SonarCloud è un servizio web che permette di eseguire e gestire dei controlli di qualità del codice presente all'interno di un repository e di riportare tutti i risultati delle analisi effettuate in un cruscotto accessibile tramite web.
È possibile integrarlo direttamente all'interno del servizio di versionamento utilizzato per il codice di programmazione; nel nostro caso, risulta completamente integrato con Git e permette di determinare, staticamente, possibili problemi del codice scritto. In particolare:

- può essere utilizzato congiuntamente a Github, in modo da effettuare automaticamente la verifica delle regole impostate per il progetto contenuto all'interno dei repository presenti;
- se integrato con Github, permette di eseguire i controlli ad ogni pull request verso il branch principale e di bloccare l'operazione se il nuovo codice non rispetta le regole impostate;
- permette di rilevare potenziali bug, vulnerabilità e *code smells* presenti nel codice, segnalandoli opportunamente all'interno del cruscotto web;
- se è stato configurato uno strumento di controllo del *code coverage*, il report effettuato in seguito all'esecuzione di test di unità può essere riportato a cruscotto, assieme ai risultati dell'analisi statica.

SonarLint

- **componenti software:** swe-api, swe-kafka-db, swe-gateway, swe-telegram, swe-webapp;
- **linguaggio:** tutti.

SonarLint è un plugin della suite di SonarCloud che permette di eseguire dei controlli di qualità del codice direttamente durante la scrittura.
È possibile installarlo direttamente all'interno dell'IDE in cui si scrive codice di programmazione; nel nostro caso, risulta completamente integrato con tutta la suite di JetBrains (IntelliJ Idea, PHPStorm e WebStorm) e permette di determinare, staticamente, possibili problemi del codice scritto. In particolare:

- può essere utilizzato congiuntamente a SonarCloud, in modo da effettuare automaticamente il bind con le regole impostate, sul servizio web, per il progetto analizzato;
- permette di rilevare potenziali bug, vulnerabilità e *code smells* presenti nel codice, segnalandoli opportunamente all'interno dell'IDE utilizzato.

3.5.6.3 Analisi dinamica del codice

JUnit

- **componenti software:** swe-api, swe-kafka-db, swe-gateway;
- **linguaggio:** Java.

JUnit è un framework, integrato dal gestore di dipendenze *Maven*, che permette di eseguire automaticamente i test di unità per codice Java.
Permette di essere configurato direttamente all'interno del file `pom.xml` presente nella cartella di progetto e, ad ogni build del prodotto software, esegue automaticamente tutti i test definiti nel progetto, segnalando quelli che non hanno avuto esito positivo, tramite degli opportuni errori.
Non necessita di particolati configurazioni in quanto rileva automaticamente i test disponibili, tramite l'apposita cartella *test* inserita all'interno del progetto, purché rispettino le convenzioni di nomenclatura ed utilizzino opportunamente i metodi e le annotazioni definite dal framework per i test di unità.
In particolare:

- una classe utilizzata per eseguire dei test di unità, su di una classe del progetto, deve avere lo stesso nome della classe testata, seguito dalla parola `Test`;
- per definire un metodo che implementi un test, questo deve essere marcato con l'annotazione `@Test`;
- i nomi dei metodi di test devono essere parlati ed autoesplicativi del tipo di test che viene eseguito.

PHPUnit

- **componenti software:** swe-webapp;

- **linguaggio:** PHP.

PHPUnit è un framework, integrato dal gestore di dipendenze *Composer*, che permette di eseguire automaticamente i test di unità per codice PHP.
Permette di essere configurato tramite un apposito file chiamato `phpunit.xml` (il quale deve essere versionato), presente nella cartella di progetto e, tramite l'invocazione di uno specifico comando da terminale, consente l'esecuzione automatica di tutti i test definiti nel progetto, segnalando quelli che non hanno avuto esito positivo, tramite degli opportuni errori.
Tramite il file di configurazione è possibile specificare sia la cartella in cui sono definiti i test da eseguire, sia i file da considerare nel calcolo del *code coverage*, sia il percorso in cui salvare il file con il report del coverage calcolato in seguito all'esecuzione dei vari test.
Inoltre, come *JUnit*, definisce delle convenzioni di nomenclatura e delle classi apposite per la definizione dei test. In particolare:

- tutte le classi che implementano dei test di unità devono essere contenute all'interno della cartella `tests`, presente all'interno del progetto;
- una classe utilizzata per eseguire dei test di unità, su di una classe del progetto, deve avere lo stesso nome della classe testata, seguito dalla parola `Test`;
- per definire una classe che implementi dei test, questa deve essere estesa dalla classe `TestCase`, fornita dal framework;
- i nomi dei metodi di test devono essere parlati ed autoesplicativi del tipo di test che viene eseguito.

Jest

- **componenti software:** swe-telegram, swe-webapp;
- **linguaggio:** JavaScript, Vue.js.

Jest è un framework, integrato dal gestore di dipendenze *NPM*, che permette di eseguire automaticamente i test di unità per codice JavaScript.
Permette di essere configurato tramite un'apposita sezione all'interno del file `package.json`, presente nella cartella di progetto e, tramite l'invocazione di uno specifico comando da terminale, consente l'esecuzione automatica di tutti i test definiti nel progetto, segnalando quelli che non hanno avuto esito positivo, tramite degli opportuni errori.
Tramite le opzioni di configurazione è possibile specificare sia la cartella in cui sono definiti i test da eseguire, sia i file da considerare nel calcolo del *code coverage*, sia il percorso in cui salvare il file con il report del coverage calcolato in seguito all'esecuzione dei vari test.
Inoltre, come *JUnit* e *PHPUnit*, definisce delle convenzioni di nomenclatura e delle classi apposite per la definizione dei test. In particolare:

- tutte le classi che implementano dei test di unità devono essere contenute all'interno della cartella `__tests__`, presente all'interno del progetto;
- una classe utilizzata per eseguire dei test di unità, su di una classe del progetto, deve avere lo stesso nome della classe testata, seguito dalla parola `.test`;

- per definire un metodo che implementi dei test, questa deve essere implementato all'interno della chiamata al metodo `test()`, fornito dal framework;
- per rendere parlati ed autoesplicativi del tipo di test che viene eseguito, i metodi devono essere accompagnati da un'opportuna stringa di testo che li descriva.

JaCoCo

- **componenti software:** swe-api, swe-kafka-db, swe-gateway;
- **linguaggio:** Java.

JUnit è un plugin, integrato dal gestore di dipendenze *Maven*, che permette di eseguire automaticamente il report del *code coverage* in seguito all'esecuzione dei test di unità per codice Java.
Permette di essere configurato direttamente all'interno del file `pom.xml` presente nella cartella di progetto e, ad ogni build del prodotto software, esegue automaticamente il report per la copertura garantita da tutti i test eseguiti sui file Java del progetto.
Non necessita di particolati configurazioni in quanto rileva automaticamente i test eseguiti e i file Java del progetto su cui effettuare il controllo di *code coverage*.
Inoltre, per agevolare la fruizione del risultato dell'analisi eseguita sul progetto, è possibile integrare il report con uno servizio di visualizzazione dettagliata che faccia riferimento all'intero repository e ne tenga anche lo storico. Nel caso del progetto, i dati raccolti sono inviati al progetto presente su *SonarCloud*, in modo da centralizzare tutte le informazioni riguardanti la verifica del software presente nel repository.

Coveralls

- **componenti software:** swe-telegram, swe-webapp;
- **linguaggio:** JavaScript, Vue.js.

Coveralls è un servizio web che permette di visualizzare e gestire dati di *code coverage* riguardanti codice presente all'interno di un repository e di riportare tutti i risultati delle analisi effettuate in un cruscotto accessibile tramite web.
È possibile integrarlo direttamente all'interno del servizio di versionamento utilizzato per il codice di programmazione; nel nostro caso, risulta completamente integrato con Github e permette di gestire tutto lo storico delle analisi di copertura eseguite sul codice presente all'interno del repository.
In particolare, il servizio non si occupa attivamente di effettuare il controllo sul codice, bensì riceve i report forniti da altri strumenti di analisi e, tramite un apposito token, li associa al corretto repository mostrando la variazione dei risultati nel tempo.
Il suo utilizzo è stato necessario in quanto non è stato possibile integrare i report di copertura per i *submodules* swe-telegram e swe-webapp direttamente all'interno di *SonarCloud*, questo a causa della diversa modalità di analisi del codice JavaScript e PHP, rispetto a quello Java, da parte di quest'ultimo servizio web.

3.6 Validazione

3.6.1 Scopo

Lo scopo del processo di validazione consiste nel garantire che il prodotto rispetti le richieste del committente, e quindi che esegua correttamente.

3.6.2 Aspettative

Per garantire che venga raggiunto lo scopo, le attività vengono testate dal processo di validazione, mandandole in esecuzione e confrontando l'output ottenuto con quello atteso.

3.6.3 Descrizione

Il processo di validazione esegue il test completo sul sistema affinché sia chiaro se sono state rispettate le necessità del committente, il che porta a definire se il prodotto esegue in modo corretto. Per poter effettuare questo processo è necessario che venga eseguito dopo il processo di verifica, in modo che tutte le unità del sistema permettano il test completo su di esso.

3.6.4 Attività

3.6.4.1 Test

In questo processo viene eseguito il test di accettazione, effettuato insieme al committente per verificare se il prodotto rispetta le richieste. Per elencare le specifiche del test si è scelta una rappresentazione tabellare contenente il codice del componente da testare, la descrizione dei test, il suo stato di avanzamento e il risultato del test stesso secondo la seguente nomenclatura:
Stato:

- **I:** il test è stato implementato;
- **NI:** il test non è ancora stato implementato.

Risultato:

- **S:** il test ha successo;
- **F:** il test ha fallito.

Tuttavia si è scelto di omettere il risultato dei test in quanto non significativo allo stato attuale del prodotto.

Test di Accettazione

I test di accettazione (o collaudo) accertano il soddisfacimento degli use case e dei requisiti concordati con il cliente. Più in particolare i test di accettazione servono a confermare che i requisiti derivati dai

casi d'uso specificati nel capitolato siano stati soddisfatti. Questi test richiedono perciò la presenza del committente e del proponente. Per classificare questi tipi di test verrà associata un codice ad ognuno di essi secondo il seguente modello:

TA-[Identificativo]

Dove:

- **Identificativo:** è un numero progressivo il cui obiettivo sarà di contraddistinguere il singolo requisito da testare; parte da 1.

3.6.5 Metriche

Al momento, per il processo di validazione non sono state individuate particolari metriche di qualità.

3.6.6 Strumenti

Non sono stati individuati degli strumenti per il processo di validazione.

4 Processi organizzativi

4.1 Gestione dei processi

4.1.1 Scopo

Il processo di gestione dei processi ha lo scopo di:

- identificare i possibili rischi e definirne la gestione;
- definire un modello di sviluppo;
- pianificare i task da svolgere in base alle scadenze temporali;
- calcolare un preventivo in termini di ore e costi suddiviso per ruoli;
- calcolare un preventivo a finire delle risorse richieste, visto il consuntivo di periodo.

4.1.2 Aspettative

Le principali aspettative del processo di gestione dei processi sono:

- agire preventivamente per evitare i rischi identificati e, qualora si verificassero, limitare le loro ripercussioni sull'efficacia e l'efficienza del lavoro svolto dal gruppo;
- effettuare una pianificazione ragionevole dei task da svolgere ed assegnarli in modo equilibrato ai diversi ruoli, tenendo conto dei tempi e delle risorse disponibili;
- gestire i componenti del gruppo e i loro task in modo da facilitare la collaborazione e la comunicazione interna tra di loro;
- mantenere sotto controllo l'andamento del progetto, monitorando il lavoro svolto dal gruppo in modo tale da non comprometterne l'efficienza.

4.1.3 Descrizione

Le attività previste dal processo di gestione dei processi sono raccolte nel PIANO DI PROGETTO v2.5.0+b1.0$_D$, la cui redazione è in carico al responsabile, con la collaborazione dell'amministratore.
Nello specifico, sono trattati:

- analisi dei rischi, classificazione degli stessi;
- istanziazione dei processi che realizzano il modello di sviluppo adottato;
- pianificazione dei task e assegnazione degli stessi ai ruoli di progetto;
- stima dei costi in termini di tempo e risorse;
- calcolo delle risorse necessarie per terminare il progetto, visto il bilancio del lavoro svolto nel periodo;
- revisione delle attività sulla base dei riscontri dei rischi.

4.1.4 Attività

4.1.4.1 Inizializzazione e campo d'applicazione

Il responsabile e l'amministratore devono stabilire la fattibilità del processo controllando che le risorse a disposizione siano disponibili, adeguate ed appropriate.

4.1.4.2 Pianificazione

Il responsabile deve preparare i piani per l'esecuzione dei processi; deve suddividere i ruoli e l'assegnazione oraria per i membri del gruppo, garantendo che ogni componente assuma, nel corso del progetto, almeno una volta ogni ruolo. I ruoli richiesti dal progetto sono i seguenti:

Responsabile

Il responsabile accentra tutte le responsabilità di pianificazione, controllo, gestione e coordinamento di attività e risorse all'interno del progetto. Inoltre svolge la funzione di intermediario verso le figure esterne al gruppo, quali committente e proponente del capitolato, ed è il responsabile ultimo dei risultati del progetto.
In particolare si occupa di:

- elaborare ed emanare piani e scadenze;
- approvare l'emissione dei documenti;
- coordinare le attività, le risorse e i componenti del gruppo;
- gestire le criticità incontrate dal gruppo;
- redigere l'ORGANIGRAMMA$_G$ e il piano di progetto;
- approvare l'offerta sottoposta al committente.

Amministratore

L'amministratore è incaricato della gestione dell'ambiente di lavoro.
All'interno del gruppo egli:

- è responsabile dell'efficacia e dell'efficienza dell'ambiente di sviluppo e di tutte le installazioni di supporto;
- è responsabile della redazione ed attuazione dei piani e delle procedure per la gestione della qualità;
- controlla le versioni e le configurazioni del prodotto;
- gestisce la documentazione del progetto;
- collabora alla redazione del piano di progetto;
- redige le norme di progetto.

Analista

L'analista è il responsabile di tutte le attività di analisi svolte durante l'analisi dei requisiti, al cui termine hanno fine anche tutti i suoi incarichi all'interno del gruppo. Egli infatti è una figura che non è presente all'interno del gruppo per tutta la durata del progetto.
L'analista ha il compito di:

- studiare il dominio applicativo del progetto;
- definire i requisiti del progetto;
- redigere lo studio di fattibilità e l'analisi dei requisiti.

Progettista

Il progettista è il responsabile di tutte le attività di progettazione svolte durante la progettazione dell'architettura e la progettazione di dettaglio.
Il progettista deve:

- prendere decisioni riguardanti gli aspetti tecnici del progetto, favorendo l'efficacia e l'efficienza;
- definire l'architettura del prodotto da sviluppare, perseguendo la sua efficienza, efficacia e manutenibilità, tramite l'utilizzo di apposite tecnologie individuate a partire dai requisiti definiti dall'analista;
- redigere la specifica tecnica, la definizione di prodotto e la parte pragmatica del piano di qualifica.

Programmatore

Il programmatore è il responsabile di tutte le attività di codifica effettuate per lo sviluppo del progetto. In particolare, il programmatore è responsabile:

- dell'implementazione della specifica tecnica redatta dal progettista;
- della codifica mirata alla realizzazione del prodotto;
- della codifica di componenti di ausilio necessarie per l'esecuzione delle prove di verifica e validazione.

Verificatore

Il verificatore è il responsabile di tutte le attività di verifica dei documenti e del codice scritti dagli altri componenti del gruppo. Il suo compito è trovare errori, di qualunque tipo, nei prodotti che controlla e di segnalare tali errori a chi ha la responsabilità diretta sul quel prodotto, in modo che possa apportare le dovute correzioni.
Il verificatore non ha il compito di correggere gli errori rilevati; deve quindi:

- esaminare i prodotti in fase di revisione, utilizzando le tecniche e gli strumenti definiti nelle norme di progetto;

- indicare eventuali errori riscontrati nel prodotto in esame;
- segnalare eventuali errori rilevati al responsabile dell'oggetto in fase di verifica, in modo che possa correggerli.

4.1.4.3 Gestione delle comunicazioni

Le comunicazioni vengono gestite sfruttando tutte le potenzialità di SLACK$_G$, il quale permette di creare un *workspace* con degli appositi canali di comunicazione. Si usano i seguenti canali e prefissi:

- `#announcement`: canale di annunci importanti;
- `#offtopic`: canale di svago per parlare di argomenti extra-lavorativi;
- `#gen_[*]`: canali generali per il software e la documentazione;
- `#bot_[*]`: canali dedicati per le notifiche di GITHUB$_G$ e di GOOGLE$_G$;
- `#int_[*]`: canali specifici per i documenti interni di progetto;
- `#ext_[*]`: canali specifici per i documenti esterni di progetto;
- `#sw_[*]`: canali specifici per un componente software;
- `#devops_[*]`: canali di notifica per le operazioni di DEVOPS$_G$ per i documenti interni di progetto.

Per organizzare le riunioni si fa uso di **Google Calendar**, grazie a cui è possibile fissare degli appuntamenti sul calendario comune di tutti, notificando o meno la propria presenza. In modo più informale, infine, si fa anche uso di una chat di **Telegram** per coordinarsi su eventuali cambi di luogo o per discutere di argomenti extra-lavorativi.

Comunicazioni interne

Le comunicazioni interne riguardano i soli membri del gruppo e sono volte a favorire la collaborazione e la suddivisione del lavoro tra i componenti stessi.
Sono state individuate due categorie principali di comunicazioni:

- **relative agli incontri:** finalizzate ad organizzare gli incontri interni, in base alle necessità e alle disponibilità dei singoli membri del gruppo.
  Come strumento è stato scelto TELEGRAM$_G$, tramite il quale è stato creato un gruppo in cui ogni componente può interagire con gli altri comunicando le proprie disponibilità;
- **relative al lavoro:** finalizzate alla discussione dei task da svolgere e delle criticità incontrate durante la loro esecuzione.
  Come strumento di comunicazione è stato scelto SLACK$_G$, tramite il quale è stato creato un WORKSPACE$_G$ del gruppo al cui interno sono stati predisposti diversi canali separati, uno per ogni task in svolgimento. Ogni membro del gruppo può scrivere in ognuno dei canali presenti ed ottenere informazioni riguardanti lo stato di progresso del lavoro e le eventuali problematiche riscontrate, facilitando quindi la collaborazione.

Comunicazioni esterne

Le comunicazioni esterne riguardano, oltre ai membri del gruppo, anche il committente e l'azienda proponente, la quale si interfaccia con il gruppo tramite un suo referente interno. L'interazione con le figure esterne è di sola competenza del responsabile, il quale provvederà a tenere costantemente aggiornato il gruppo riguardo gli sviluppi del suo operato.
Lo strumento predefinito per le comunicazioni esterne è la posta elettronica, per la quale il gruppo ha previsto una casella unica all'indirizzo redroundrobin.site@gmail.com, accessibile in lettura da parte di tutti i componenti e in scrittura solamente da parte del responsabile.

Oltre alla posta elettronica, per favorire una riscontro più rapido e diretto con il proponente, quest'ultimo ha predisposto un WORKSPACE$_G$ SLACK$_G$ con, all'interno, un apposito canale riservato alle comunicazioni con il gruppo. Di conseguenza il responsabile ha provveduto alla creazione di un singolo account SLACK$_G$ del gruppo, utilizzato unicamente per comunicare con il proponente all'interno del canale. Per ridurre il carico di lavoro del responsabile è stato scelto di permettere l'accesso all'account sopra citato da parte di ciascun membro del gruppo, in modo da facilitare la fruizione degli argomenti trattati con il proponente. Tuttavia la partecipazione attiva al canale è permessa soltanto al responsabile.

4.1.4.4 Gestione degli incontri

In caso di necessità, il responsabile può organizzare delle riunioni volte a trattare argomenti critici, che richiedono un confronto diretto tra tutti i componenti del gruppo ed, eventualmente, con il proponente.
Ogni incontro dovrà essere fissato in accordo con tutti i partecipanti, in base alle loro disponibilità. Per semplificarne la gestione, il responsabile deve creare un apposito evento su Google Calendar, tramite l'account del gruppo, con tutte le informazioni riguardanti la riunione. L'evento deve poi essere condiviso con tutti i componenti del gruppo, in modo che ne possano prendere visione ed essere immediatamente informati di eventuali modifiche tramite il servizio email di notifica di Google.

Incontri interni

Agli incontri interni partecipano solamente i membri del gruppo, i quali sono tenuti a farsi trovare nel luogo prestabilito entro l'orario indicato.
Sono tollerati ritardi e assenze giustificate se segnalati con sufficiente anticipo da permettere al responsabile di apportare le dovute modifiche al programma della riunione, e a tutti gli altri partecipanti di prenderne visione.
Gli incontri interni possono avvenire in due modalità:

- **fisica:** incontri svolti di persona, nei quali i membri del gruppo si ritrovano per discutere di tematiche critiche e prendere decisioni in merito a questioni importanti che riguardano il lavoro da svolgere;
- **virtuale:** incontri svolti tramite chiamate di gruppo, nei quali i componenti possono discutere tra loro di eventuali dubbi sorti durante lo svolgimento dei task assegnati o comunicare le loro difficoltà nel portare avanti uno specifico compito, in modo che possano essere attuate le opportune misure correttive nel più breve tempo possibile.
  Come strumento di comunicazione a chiamate è stato scelto DISCORD$_G$, che permette, tramite la predisposizione di un apposito server del gruppo, di realizzare una suddivisione a canali

vocali degli argomenti da trattare nelle diverse discussioni. Questi canali sono del tutto analoghi, per funzionamento, a quelli messi a disposizione da SLACK$_G$, con l'unica differenza che la comunicazione al loro interno avviene tramite chiamate vocali.

Incontri esterni

Negli incontri esterni, assieme ai membri del gruppo, sono coinvolti anche uno o più rappresentanti dell'azienda proponente.
Data la presenza di persone esterne, questi incontri assumono una criticità maggiore, che deve essere opportunamente gestita dal responsabile tramite un'adeguata gestione delle comunicazioni che punti a mediare tra l'interno del gruppo e gli individui esterni.
Data la complessità dell'organizzazione, ritardi e assenze da parte dei componenti del gruppo sono da evitare, a meno di un'opportuna giustificazione non preventivabile.
Gli incontri esterni, come quelli interni, possono avvenire in due modalità:

- **fisica:** incontri svolti di persona presso la sede dell'azienda proponente, nei quali i membri del gruppo si ritrovano assieme ai rappresentanti della stessa, per avere un riscontro diretto sulle decisioni prese in merito al lavoro da svolgere e per chiarire eventuali parti del progetto che non sono del tutto chiare al gruppo;
- **virtuale:** incontri svolti tramite chiamate di gruppo, nei quali i componenti possono discutere con il proponente di eventuali dubbi sorti durante l'analisi e/o lo svolgimento dei task assegnati, in modo da poter porre domande dirette ed aver una risposta immediata da parte dell'azienda. Come strumento di comunicazione a chiamate è stato scelto Skype, che permette di effettuare facilmente videochiamate con una o più persone.

Verbali degli incontri

Successivamente ad un incontro, sia esso interno od esterno, il segretario incaricato dal rappresentate ne redige il verbale.
La nomenclatura dei documenti dei verbali deve essere tale da permettere l'identificazione univoca di ogni verbale e il loro ordinamento temporale all'interno del REPOSITORY$_G$. A questo scopo la forma da adottare è la seguente:

V[T]_[AAAA-MM-GG]_[Numero]

Dove:

- **V:** indica che si tratta di un verbale;
- **T:** indica la tipologia del verbale, ossia:
  - **I:** indica che il verbale si riferisce ad un incontro interno;
  - **E:** indica che il verbale si riferisce ad un incontro esterno.
- **AAAA-MM-GG:** indica la data di svolgimento dell'incontro a cui il verbale si riferisce;
- **Numero:** numero intero progressivo che fornisce un'indicazione riguardo all'ordine temporale di svolgimento degli incontri.

4.1.4.5 Gestione degli strumenti di coordinamento

In ogni momento, durante lo sviluppo del progetto, tutti i membri del gruppo devono poter sapere quali compiti sono pianificati, quali sono in corso e quali sono già stati completati. Inoltre, ognuno deve essere a conoscenza dei compiti a lui assegnati e della relativa data di scadenza per il loro completamento, in modo da poter gestire il proprio carico di lavoro. Infine, il responsabile deve poter assegnare nuovi compiti ai membri del gruppo e controllarne lo stato di avanzamento per verificarne la coerenza con la pianificazione.
Per soddisfare queste necessità è stato scelto di utilizzare l'ISSUE TRACKING SYSTEM$_G$ fornito da GITHUB$_G$ per il REPOSITORY$_G$ usato dal gruppo per il VERSIONAMENTO$_G$ del progetto.
Questo strumento permette di creare delle ISSUE$_G$ che possono rappresentare dei singoli compiti, assegnabili ad una o più persone, attraverso le seguenti informazioni:

- **titolo:** nome del compito da eseguire;
- **descrizione:** descrizione dettagliata del compito da eseguire;
- **assegnatari:** persone a cui compete lo svolgimento del compito;
- **bacheca:** cruscotto di progetto in cui il compito sarà monitorato;
- **scadenza:** data entro la quale lo svolgimento del compito deve essere completato.

Ogni ISSUE$_G$ attraversa degli stati che permettono di monitorare l'avanzamento nello svolgimento del compito che essa rappresenta, questi sono:

- **to do:** compito da svolgere;
- **in progress:** compito in svolgimento;
- **done:** compito svolto.

È stato scelto l'ISSUE TRACKING SYSTEM$_G$ di GITHUB$_G$ in quanto è completamente integrato con il REPOSITORY$_G$ del progetto, è semplice da gestire e si adatta a più casi di utilizzo.

4.1.4.6 Gestione dei rischi

I rischi che si verificano durante lo svolgimento del progetto devono essere prontamente rilevati, classificati e documentati nel *Piano di Progetto*, prevedendo, inoltre, delle strategie per la loro gestione. Una volta fatto ciò, oltre a individuare nuovi rischi, è necessario monitorare i quelli noti, attuando le corrette strategie di mitigazione nel caso in cui essi si verifichino nuovamente.
Per permettere un facile riferimento ad ogni rischio documentato, è stata prevista una codifica che, oltre a consentirne l'identificazione univoca, riesca a mostrarne anche le principali caratteristiche. I codici di riferimento dei rischi hanno, quindi, la seguente struttura:

RSK-[Tipo][Probabilità][Gravità]-[ID]

dove:

- **RSK:** indica che si tratta di un rischio;
- **Tipo:** tipologia del rischio, che può essere:
  - **O:** organizzativo, riguardante l'organizzazione del lavoro all'interno del gruppo e la gestione dei costi e delle risorse, tra cui persone e tempi;
  - **P:** personale, riguardante le persone che fanno parte del gruppo, le loro conoscenze e le loro capacità;
  - **R:** requisiti, riguardante le modifiche ai requisiti che possono avvenire durante lo sviluppo del progetto;
  - **S:** strumentale, riguardante gli strumenti software impiegati per lo sviluppo del progetto, il loro utilizzo e le loro prestazioni;
  - **T:** tecnologico, riguardante le tecnologie hardware o software impiegate per lo sviluppo del progetto, il loro utilizzo e le loro funzionalità.
- **Probabilità:** livello di possibilità che il rischio si verifichi, che può essere:
  - **A**: alta;
  - **M**: media;
  - **B**: bassa.
- **Gravità:** livello di pericolosità del rischio, che può essere:
  - **A:** accettabile;
  - **T:** tollerabile;
  - **I:** inaccettabile.
- **ID:** numero intero progressivo che fornisce la numerazione dei vari rischi della stessa tipologia, probabilità e gravità.

4.1.4.7 Esecuzione e controllo

L'amministratore deve monitorare l'esecuzione dei processi, fornendo sia report interni per quanto riguarda lo stato di avanzamento del progetto, che report esterni verso il committente. L'amministratore dovrà inoltre investigare, analizzare e risolvere i problemi riscontrati durante l'esecuzione dei processi, assicurandosi che l'impatto di ciascun cambiamento sia controllato, monitorato e documentato.

4.1.4.8 Conclusione

L'amministratore dovrà infine determinare se il processo in discussione possa essere considerato come concluso, documentando gli eventuali risultati forniti dalla strumentazione per quanto riguarda la completezza.

4.1.5 Metriche

4.1.5.1 QP-1 Gestione dei processi

Scopo

Si vuole gestire la copertura di risorse disponibili per la realizzazione del progetto, monitorando i costi aggiuntivi e le tempistiche non rispettate dalla schedulazione pianificata. Questo può essere utile al cliente per capire in fase di sviluppo l'andamento del progetto a livello di gestione delle risorse.

Introduzione alle Metriche

Per la gestione delle risorse si farà uso delle seguenti metriche:

- QM-PROC-1 Budgeted cost of work scheduled (BCWS);
- QM-PROC-2 Actual cost of work performed (ACWP);
- QM-PROC-3 Budgeted cost of work performed (BCWP);
- QM-PROC-4 Schedule variance (SV);
- QM-PROC-5 Cost variance (CV).

QM-PROC-1 Budgeted cost of work scheduled (BCWS)

- **Descrizione:** la metrica BCWS definisce il costo pianificato per realizzare le attività di progetto alla data corrente;
- **Unità di misura:** il costo pianificato è misurato in EURO.

QM-PROC-2 Actual cost of work performed (ACWP)

- **Descrizione:** la metrica ACWP definisce il costo effettivamente sostenuto per realizzare le attività di progetto alla data corrente;
- **Unità di misura:** il costo sostenuto è misurato in EURO.

QM-PROC-3 Budgeted cost of work performed (BCWP)

- **Descrizione:** la metrica BCWP definisce il valore delle attività realizzate alla data corrente. In altre parole, misura il valore del prodotto fino ad ora realizzato;
- **Unità di misura:** il valore del prodotto è misurato in EURO.

QM-PROC-4 Schedule variance (SV)

- **Descrizione:** la metrica SV indica se si è in anticipo, in ritardo o in linea rispetto alle schedulazioni pianificate per il progetto. Questo può essere utile per il cliente per valutare l'efficacia del gruppo nei confronti della realizzazione del progetto;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula per il calcolo della metrica è la seguente:

$$\text{SV} = \frac{\text{BCWP} - \text{BCWS}}{\text{BCWS}} \times 100$$

- **Risultato:**
  - un risultato **positivo** (> 0) indica che il progetto è avanti rispetto alla schedulazione;
  - un risultato **negativo** (< 0) indica che il progetto è indietro rispetto alla schedulazione;
  - un risultato **pari a zero** indica che il progetto è in linea rispetto alla schedulazione.

QM-PROC-5 Cost variance (CV)

- **Descrizione:** la metrica CV indica se il valore del costo realmente maturato è maggiore, minore o uguale rispetto al costo effettivo. In altre parole, permette di comprendere con che livello di efficienza il gruppo sta sviluppando il progetto, rispetto a quanto pianificato;
- **Unità di misura:** la metrica viene espressa in percentuale;
- **Formula:** la formula per il calcolo della metrica è la seguente:

$$\text{CV} = \frac{\text{BCWP} - \text{ACWP}}{\text{BCWP}} \times 100$$

- **Risultato:**
  - un risultato **positivo** (> 0) indica che il progetto sta sviluppando con un costo minore rispetto a quanto pianificato (maggiore efficienza);
  - un risultato **negativo** (< 0) indica che il progetto sta sviluppando con un costo maggiore rispetto a quanto pianificato (minore efficienza);
  - un risultato **pari a zero** indica che il progetto sta sviluppando con un costo in linea rispetto a quello pianificato.

4.1.5.2 QP-2 Gestione dei rischi

Per la gestione dei rischi si farà uso delle seguenti metriche:

- QM-PROC-6 Unbudgeted risks (UR).

QM-PROC-6 Unbudgeted risks (UR)

- **Descrizione:** la metrica UR viene utilizzata per tracciare in modo incrementale tutti i nuovi rischi non precedentemente preventivati che si presentano durante una fase del progetto;
- **Unità di misura:** la metrica viene espressa con un valore intero che parte da 0;
- **Formula:** per ogni rischio non preventivato e non individuato precedentemente che viene rilevato, si incrementa di una unità il numero di rischi rilevati fino alla data corrente, a partire da una fase del progetto. La formula della metrica è la seguente:

$$\mathrm{UR} = \mathrm{UR} + 1$$

- **Risultato:**
  - un valore pari a 0, indica che non sono stati trovati rischi nella fase del progetto;
  - un valore superiore a 0, indica che sono stati trovati rischi nella fase del progetto.

4.1.6 Strumenti

Durante lo svolgimento del progetto, per favorire l'attuazione delle procedure descritte in precedenza, il gruppo ha scelto di utilizzare i seguenti strumenti:

- **Telegram$_G$:** strumento di messaggistica utilizzato per le comunicazione relative agli incontri tra i membri del gruppo;
- **Slack$_G$:** strumento di collaborazione utilizzato dal gruppo sia per le comunicazioni interne relative al lavoro svolto dai singoli componenti, sia per le comunicazioni con il proponente;
- **GMail:** servizio di posta elettronica fornito da Google, tramite l'account del gruppo, attraverso il quale avvengono le comunicazioni con il committente ed il primo contatto con il proponente;
- **Google Calendar:** calendario fornito da Google, tramite l'account del gruppo, sul quale vengono segnati tutti gli incontri programmati in modo che siano visibili a tutti i componenti del gruppo;
- **Discord$_G$:** applicativo gratuito che offre la possibilità di creare dei canali vocali e testuali per svolgere degli incontri virtuali con il gruppo;
- **Skype:** strumento che offre un servizi di chiamata e videochiamata online, utilizzato per gli incontri virtuali con il proponente;
- **GitHub$_G$:** strumento che fornisce un servizio di versionamento remoto, utilizzato per il salvataggio, la storicizzazione e la condivisione di tutti i file del progetto;
- **Issue Tracking System$_G$:** strumento fornito da GitHub$_G$, utilizzato per la gestione e il coordinamento dei compiti da svolgere.

4.2 Formazione del personale

4.2.1 Scopo

I membri del gruppo Red Round Robin sono tenuti a formarsi individualmente sulle tecnologie richieste per il completamento del progetto e, nel caso di incomprensioni o problemi nell'utilizzo di esse, il proponente si dichiara disponibile a corsi di formazione e chiarimenti.

4.2.2 Descrizione

Ogni persona dovrà documentarsi sui linguaggi e gli strumenti di programmazione che verranno utilizzati per tutto il periodo di sviluppo del progetto, se sconosciuti.

4.2.3 Attività

4.2.3.1 Materiale per la formazione

In questa sezione sono elencate le tecnologie e i linguaggi di programmazione necessari per lo svolgimento del capitolato con i relativi link al sito ufficiale.

Linguaggi di programmazione

- **LaTeX**: www.latex-project.org;
- **Java**: www.java.com;
- **PHP**: www.php.net;
- **HTML e CSS**: www.w3schools.com.

Tecnologie

- **GitHub**: github.com;
- **Spring Framework (Java)**: spring.io;
- **Apache Maven**: maven.apache.org;
- **Apache Kafka**: kafka.apache.org;
- **Bootstrap**: www.getbootstrap.com;
- **Laravel 6.x**: laravel.com;
- **Docker**: docker.com;
- **PostgreSQL**: www.postgresql.org;
- **TimescaleDB**: timescale.com.

4.2.4 Metriche

Per la formazione del personale non si fa uso di particolari metriche di qualità.

4.2.5 Strumenti

Non sono stati individuati degli strumenti che accompagnino questo processo.