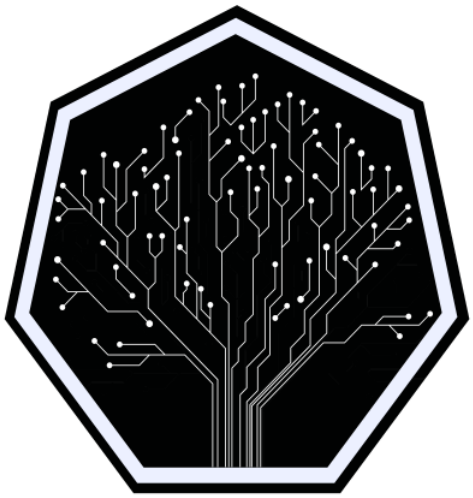

VRAM SOFTWARE

Progetto "Predire in Grafana"

Norme di Progetto

02 marzo 2020

Versione	4.1.1
Approvazione	Rampazzo Marco
Redazione	Dalla Libera Marco
	Spreafico Alessandro
Verifica	Schiavon Rebecca
	Corrizzato Vittorio
	Santagiuliana Vittorio
	Toffoletto Massimo
Stato	Approvato
Uso	Interno
Destinato a	Zucchetti
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Email di riferimento	vram.software@gmail.com

Descrizione

Questo documente contiene gli strumenti e le norme adottate dal gruppo *VRAM Software*

Registro delle modifiche

Versione	Data	Nominativo	Ruolo	Descrizione
4.1.1	2020-03-02	Rampazzo Marco	*Responsabile di progetto*	Approvazione documento.
3.2.4	2020-03-01	Spreafico Alessandro e Santagiuliana Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.5, §2.2.5, §3.2.5, §3.3.5 e §3.4.6.
3.2.3	2020-02-28	Spreafico Alessandro e Schiavon Rebecca	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §B e §C.
3.1.2	2020-02-20	Dalla Libera Marco e Santagiuliana Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.5, §2.2.5, §3.2.5, §3.3.5 e §3.4.6.
3.1.1	2020-02-16	Rampazzo Marco	*Responsabile di progetto*	Approvazione documento.
2.3.2	2020-02-14	Spreafico Alessandro e Toffoletto Massimo	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.5, §2.2.5, §3.2.5, §3.3.5 e §3.4.6.
2.2.2	2020-02-12	Spreafico Alessandro e Toffoletto Massimo	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §3.3, §3.4.5, §3.6.5 e §4.1.4.
2.1.1	2020-02-08	Rampazzo Marco	*Responsabile di progetto*	Approvazione documento.
1.4.4	2020-02-06	Dalla Libera Marco e Santagiuliana Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.5, §2.2.5, §3.2.5, §3.3.5 e §3.4.6.

1.3.3	2020-02-04	Dalla Libera Marco e Schiavon Rebecca	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §2.2.4.3 e §3.6.
1.2.2	2020-02-02	Dalla Libera Marco e Corrizzato Vittorio	*Amministratore* e *Verificatore*	Correzione della struttura del documento, dei riferimenti, dello stile tipografico, del registro delle modifiche e associazione più precisa delle metriche ai relativi processi tutto come segnalato dal committente.
1.1.1	2020-01-11	Corrizzato Vittorio	*Responsabile di progetto*	Approvazione documento.
0.8.4	2020-01-11	Santagiuliana Vittorio e Schiavon Rebecca	*Amministratore* e *Verificatore*	Aggiornamento e verifica finale.
0.8.3	2020-01-06	Santagiuliana Vittorio e Schiavon Rebecca	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.4.2, §2.1.4.3, §4.1.6 e §A.
0.7.3	2020-01-02	Rampazzo Marco e Santagiuliana Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §3.3, §3.3.6 e §3.4.5.
0.6.2	2020-01-02	Rampazzo Marco e Santagiuliana Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.2.4.1 e §2.1.4.3.
0.5.2	2019-12-23	Rampazzo Marco e Spreafico Alessandro	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §2.2.4.1, §2.1.4.2, §3.1.6 e §3.2.5.

0.4.1	2019-12-04	Toffoletto Massimo e Corrizzato Vittorio	*Amministratore* e *Verificatore*	Aggiornamento e verifica dei paragrafi §2.1.4.1 e §2.2.5.
0.3.1	2019-11-23	Corrizzato Vittorio e Schiavon Rebecca	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §3.4, §3.5 e §4.
0.2.0	2019-11-23	Corrizzato Vittorio e Schiavon Rebecca	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §3.1 e §3.2.
0.1.0	2019-11-23	Corrizzato Vittorio e Schiavon Rebecca	*Amministratore* e *Verificatore*	Stesura e verifica dei paragrafi §1, §2.1.4.1, §2.1.5 e §2.2.5.

Indice

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento ha come scopo la definizione di regole e convenzioni che stanno alla base del progetto$_G$ e che tutti i membri del gruppo devono seguire. In questo modo si garantisce consistenza e omogeneità in tutto il materiale del progetto$_G$ stesso. Infatti ogni componente del gruppo è obbligato a visionare questo documento e a rispettarne rigorosamente le regole definite al fine di lavorare secondo una metodologia coesa e uniforme.

1.2 Scopo del prodotto

Il capitolato$_G$ C4 ha lo scopo di implementare un plug-in di Grafana$_G$ scritto in linguaggio Javascript che esegue gli algoritmi di Support Vector Machine SVM$_G$ o Regressione Lineare RL$_G$, i quali, leggendo da un file JSON la loro configurazione, saranno in grado di generare previsioni che potranno essere aggiunte al flusso di monitoraggio. Il plug-in prevede il superamento di determinati livelli di soglia per generare un allarme attraverso il meccanismo di Grafana$_G$ detto alert$_G$. Il plug-in utilizza un applicativo esterno per addestrare gli algoritmi di SVM$_G$ ed RL$_G$ o altri algoritmi di predizione. Dunque il plug-in permette un'attività di monitoraggio grazie alla quale gli operatori possono intervenire con ogni cognizione di causa sul sistema.

1.3 Ambiguità e glossario

Questo documento è corredato da un *Glossario v. 2.1.1* dove saranno illustrati i termini tecnici o altamente specifici per evitare ambiguità in essi. Le voci interessate sono identificate da una 'G' a pedice.

1.4 Classificazione delle metriche

In questo documento vengono definite delle metriche che garantiscono qualità di processo$_G$ e di prodotto$_G$ in modo misurabile. Per avere un riferimento univoco alle metriche che permetta di individuarle facilmente all'interno dei documenti, viene utilizzato un codice identificativo. Questo codice avrà la seguente struttura:
M[Numero]
dove

- **M** indica che si tratta di una metrica;

- **Numero** è un numero intero progressivo di due cifre.

1.5 Riferimenti

1.5.1 Riferimenti normativi

1. **Capitolato$_G$ d'appalto C4 - Predire in Grafana$_G$**: `https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C4.pdf`;

1.5.2 Riferimenti informativi

1. **Terry Rout. Information technology - Software life cycle processes, Standard ISO/IEC 12207:1995 Australia&New Zealand**
   Sezione: Primary life cycle processes.
   Sezione: Supporting life cycle processes.
   Sezione: Organizational life cycle processes: `https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf` (paragrafi §5.2, §5.3, §6.1-§6.5, §6.8, §7.1 e §7.4);
2. **Slide L05 del corso Ingegneria del Software - Ciclo di vita del software**: `https://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L05.pdf`;
3. **Slide L12 del corso Ingegneria del Software - Qualità di prodotto**: `https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L12.pdf`.

2 Processi primari

2.1 Fornitura

2.1.1 Scopo

Lo scopo del processo$_G$ di fornitura è stabilire norme, tempi e risorse necessarie a svolgere l'intero progetto$_G$. Inizialmente è necessario comprendere le richieste dei proponenti svolgendo un'analisi sui capitolati$_G$e sulla loro fattibilità. Una volta compresi, viene redatto il documento *Studio di Fattibilità* nel quale viene manifestata la decisione del capitolato$_G$ scelto. Dunque è necessario formalizzare un contratto, insieme all'azienda proponente, per la consegna del prodotto$_G$. Nel contratto bisogna definire gli obiettivi di qualità da perseguire che vengono formalizzati nel documento *Piano di Qualifica* e che dovranno essere validati al termine del progetto e una pianificazione di attività, tempi e risorse fino alla consegna del progetto$_G$ che viene formalizzata nel documento *Piano di Progetto*.

2.1.2 Aspettative

Il gruppo si aspetta di mantenere un dialogo costante con l'azienda proponente e di instaurare un rapporto collaborativo per comprendere meglio le loro esigenze. In particolare ci si focalizza sui seguenti termini:

- definizione degli elementi fondamentali su cui focalizzarsi per soddisfare le necessità del proponente;
- definizione di vincoli e requisiti sui processi$_G$;
- stima delle tempistiche e della pianificazione del lavoro;
- verifica continua e chiarimento di eventuali dubbi;
- validazione del prodotto$_G$;
- accordo sulla qualifica del prodotto$_G$ e dei processi$_G$.

2.1.3 Descrizione

La sezione dei processi$_G$ di fornitura specifica le regole che devono essere rispettate affinché il nostro gruppo possa diventare fornitore del proponente *Zucchetti*. Per fare quanto sopra descritto si devono svolgere le seguenti attività:

- avvio;

- preparazione della risposta;
- contrattazione con il proponente;
- pianificazione;
- esecuzione e controllo;
- revisione e validazione;
- completamento e consegna al proponente.

2.1.4 Attività

2.1.4.1 Avvio

Scopo

Questa attività definisce l'avvio del processo di fornitura del prodotto$_G$. Il suo scopo è analizzare i capitolati$_G$ proposti e prendere una decisione in merito a quale tra questi è stato preferito dal gruppo. Perciò risulta necessario individuare pregi e difetti di ogni capitolato$_G$ per determinare quello che bilancia meglio le aspettative e le risorse a disposizione del gruppo.

Studio di Fattibilità

Il prodotto$_G$ di questa attività è un documento redatto da un analista ed è chiamato *Studio di Fattibilità*. Al suo interno, per ogni capitolato$_G$, vengono indicati:

- **Informazioni Generali**: elenco che presenta le seguenti informazioni:
  - nome del progetto$_G$;
  - proponente;
  - committente.
- **Descrizione**: breve esposizione dell'argomento del capitolato$_G$;
- **Obiettivi di Progetto**: descrizione dettagliata centrata sugli obiettivi da raggiungere;
- **Requisiti di progetto**: elenco dei vincoli imposti dal proponente per la realizzazione del progetto$_G$;
- **Tecnologie Interessate**: elenco e descrizione sintetica delle tecnologie che dovranno essere impiegate nello svolgimento;

- **Aspetti positivi**: elenco delle principali motivazioni che porterebbero il gruppo a scegliere il capitolato$_G$;
- **Criticità e fattori di rischio**: elenco delle principali motivazioni che porterebbero il gruppo a non scegliere il capitolato$_G$;
- **Conclusioni**: sintesi delle motivazioni per cui il capitolato$_G$ è stato scelto o è stato escluso.

2.1.4.2 Preparazione della risposta

Si deve preparare una risposta alle richieste del proponente del capitolato$_G$ scelto da gruppo.

2.1.4.3 Contrattazione con il proponente

Si deve negoziare con il proponente al fine di stipulare un contratto per la fornitura del prodotto$_G$ software che sia sostenibile per entrambe le parti. Uno degli elementi fondamentali nella stipulazione del contratto è la qualità del prodotto$_G$ che vogliamo realizzare.

Rapporto con il proponente

Abbiamo definito con il proponente un rapporto di collaborazione continua per discutere assieme e capire meglio le sue necessità e come poterle soddisfarle nel modo migliore. In particolare confrontiamo gli incrementi del software che realizziamo e, nel corso del progetto, decidiamo in comune accordo quali sviluppare e con quali priorità. Infine essi devono essere validati per essere certi che soddisfino i requisiti richiesti.

2.1.4.4 Pianificazione

Scopo

Quest'attività ha lo scopo di pianificare le attività per la realizzazione del progetto$_G$ e di individuare le risorse disponibili da assegnare a ciascuna di esse. La pianificazione deve ricevere in input gli obiettivi di qualità stabiliti perché rappresentano uno degli elementi fondanti di tale attività. Inoltre deve essere fissata la descrizione di strategie e tecniche di verifica, che devono essere applicate dai verificatori nelle loro attività, e di validazione$_G$, che devono essere fatte con il proponente.

Piano di Qualifica

Un prodotto$_G$ necessario per lo svolgimento di questa attività è un documento chiamato *Piano di Qualifica*. Esso è redatto dai verificatori per la parte retrospettiva e dai progettisti per la parte programmatica. Questo documento si pone l'obiettivo di garantire la qualità di prodotto$_G$ e la qualità di processo$_G$ ed è composto da:

- **Introduzione**: descrizione generale del documento che specifica anche lo scopo del documento e del prodotto$_G$;
- **Qualità di Processo$_G$**: individuazione dei processi$_G$ da adottare in riferimento ad uno standard e delle metriche per la misurazione e il monitoraggio della loro qualità;
- **Qualità di Prodotto$_G$**: individuazione degli obiettivi posti sul prodotto$_G$ che sono necessari per raggiungere una buona qualità e delle metriche per misurarla;
- **Specifica dei test**: individuazione dei test a cui sottoporre il prodotto$_G$ per garantire che i requisiti vengano soddisfatti;
- **Standard di Qualità**: elenco degli standard di qualità scelti;
- **Resoconto delle attività di verifica**: resoconti dei risultati dati dalle metriche di qualità calcolate per ogni attività;
- **Valutazioni per il miglioramento**: esposizione dei problemi rilevati e delle soluzioni da attuare per migliorare.

Piano di Progetto

Un altro prodotto$_G$ di questa attività che rappresenta concretamente la pianificazione del lavoro da svolgere è un documento redatto dal responsabile, in collaborazione con gli amministratori, ed è chiamato *Piano di Progetto*. Esso è composto da:

- **Introduzione**: descrizione generale del documento che specifica anche lo scopo del documento e del prodotto$_G$ e il calendario delle attività;
- **Analisi dei rischi**: analisi dettagliata dei rischi che potrebbero sorgere durante il ciclo di sviluppo del progetto$_G$. In allegato vengono forniti stime probabilistiche, livelli di rischio e modalità previste per fare prevenzione o per affrontarli qualora non sia stato possibile evitarli;

- **Modello di sviluppo**: descrizione del modello di sviluppo selezionato per lo svolgimento del progetto$_G$;
- **Pianificazione**: specifica dei tempi e dei ruoli per ogni attività individuata per affrontare al meglio le scadenze del capitolato$_G$. Essa viene limitata dalla quantità di risorse;
- **Preventivo**: stima dei costi necessari per lo svolgimento di ogni scadenza prevista e conseguente costruzione del preventivo per il progetto$_G$.
- **Consuntivo di periodo**: analisi degli scostamenti nell'utilizzo delle risorse rispetto a quanto preventivato. Inoltre vengono riportate le decisioni atte a mitigare il risultato di questi scostamenti e ad evitarli in futuro per non inficiare il preventivo;
- **Riscontro dei rischi**: resoconto dell'attualizzazione dei rischi e della rispettiva manutenzione migliorativa;
- **Organigramma**: presentazione del nostro organigramma.

2.1.4.5 Esecuzione e controllo

In questa attività viene eseguito e implementato tutto ciò che è stato pianificato. Durante l'esecuzione avviene un'azione di controllo e monitoraggio continuo su quanto definito durante la pianificazione in termini di tempo, risorse e qualità. Dunque si devono identificare eventuali problemi di costi, ritardi nello svolgimento del progetto e mancati obiettivi di qualità. Tutto ciò deve essere riportato rispettivamente nei documenti *Piano di Progetto* e *Piando di Qualifica*.

2.1.4.6 Revisione e valutazione

Durante l'attività di revisione e valutazione vengono eseguite le verifiche e le validazioni del prodotto$_G$ software necessarie a dimostrare il raggiungimento dei vincoli e degli obiettivi di qualità fissati nel contratto. Questo viene fatto attraverso un resoconto che deve essere fornito al proponente.

2.1.4.7 Completamento e consegna al proponente

Quest'attività consiste nel completamento e nella consegna del prodotto$_G$ software al proponente secondo i termini specificati nel contratto.

2.1.5 Strumenti di supporto

Durante il processo$_G$ di fornitura sono utilizzati i seguenti strumenti della suite di Microsoft.

2.1.5.1 Microsoft Excel

Per svolgere semplici calcoli, creazione di grafici, diagrammi e tabelle si è scelto di utilizzare il software Microsoft Excel.

2.1.5.2 Microsoft Project

Per realizzare il diagramma di Gantt necessario per la pianificazione, in particolare di tempi, risorse e analisi dei carichi di lavoro, si è scelto di utilizzare Microsoft Project.

2.2 Sviluppo

2.2.1 Scopo

Lo scopo del processo$_G$ di sviluppo è svolgere e normare le attività necessarie per la realizzazione del prodotto$_G$ richiesto.

2.2.2 Aspettative

Il gruppo si è posto le seguenti aspettative:

- stabilire gli obiettivi di sviluppo;
- stabilire i vincoli tecnologici e di progettazione$_G$;
- realizzare un prodotto$_G$ che soddisfi i requisiti imposti dal proponente e superi le verifiche definite nel processso$_G$ di verifica per raggiungere una buona qualità.

2.2.3 Descrizione

Il processo$_G$ di sviluppo è suddiviso nelle seguenti attività:

- analisi dei requisiti;
- progettazione$_G$;
- codifica.

2.2.4 Attività

2.2.4.1 Analisi dei requisiti

Scopo

Lo scopo di questa attività è:

- individuare lo scopo del lavoro;
- individuare le richieste degli stakeholder$_G$ e metterle in relazione allo scopo del progetto$_G$;
- definire i requisiti in accordo con il proponente;
- creare i diagrammi dei casi d'uso$_G$;
- raffinare e classificare i requisiti;
- a partire dai requisiti e dai casi d'uso$_G$ fissati, procedere per raffinamenti continui per garantire un miglioramento continuo del proprio prodotto$_G$;
- tracciare i requisiti e fornire dei riferimenti per le attività di controllo dei test da fornire ai verificatori;

Descrizione

Le attività che devono essere eseguite per raggiungere gli obiettivi fissati sono:

- **Comprensione del capitolato$_G$**: eseguita attraverso la lettura e l'analisi della documentazione fornita;
- **Comunicazione con il proponente**: eseguita al fine di migliorare l'analisi del progetto$_G$;
- **Comunicazione interna**: eseguita tra i membri del gruppo;
- **Analisi dei possibili casi d'uso$_G$**.

Analisi dei Requisiti

Un prodotto di quest'attività che formalizza gli obiettivi descritti sopra è un documento chiamato *Analisi Dei Requisiti*. Esso è composto da:

- **Introduzione**: descrizione generale del documento che specifica anche lo scopo del documento e del prodotto$_G$;
- **Descrizione generale**: descrizione di obiettivi, vincoli e caratteristiche generali del prodotto$_G$;
- **Casi d'uso**: definizione degli attori e dei casi d'uso$_G$;
- **Requisiti funzionali**: definizione dei requisiti funzionali individuati;
- **Requisiti di vincolo**: definizione dei requisiti di vincolo individuati;
- **Requisiti qualitativi**: definizione dei requisiti qualitativi individuati;
- **Requisiti prestazionali**: definizione dei requisiti prestazionali individuati;
- **Tracciamento dei requisiti**: definizione del tracciamento tra fonti e requisiti e viceversa.

Classificazione dei casi d'uso

Un caso d'uso$_G$ definisce un insieme di scenari con un obiettivo finale in comune per un attore$_G$. Sono ottenuti attraverso la valutazione di ogni requisito e descrivono l'insieme delle funzionalità fornite dal sistema dal punto di vista degli utenti. Per descrivere ogni caso d'uso$_G$ viene utilizzata la seguente struttura:

- codice identificativo;
- Titolo;
- Attori$_G$ **primari**;
- Attori$_G$ **secondari**;
- Descrizione;
- Precondizione;
- Post-condizione;
- Scenario principale;

- Scenario alternativo (ove presente);
- Inclusioni (ove presenti);
- Estensioni (ove presenti);
- Generalizzazioni (ove presenti);
- Diagramma UML (ove necessario).

Il codice identificativo sarà scritto in questo formato:
UC[codice_padre].[codice_figlio]
Dove:

- **codice_padre**: numero che identifica univocamente i casi d'uso$_G$;
- **codice_figlio**: numero progressivo che identifica i sotto-casi;

Classificazione dei requisiti

Per descrivere un requisito viene utilizzata la seguente struttura:

- codice identificativo;
- classificazione;
- descrizione;
- fonti.

Codice identificativo: codice univoco scritto nel seguente formato:
R[Importanza][Tipologia][Codice]
Dove:

- **Importanza**: può assumere i seguenti valori:
  - 1: requisito obbligatorio;
  - 2: requisito desiderabile;
  - 3: requisito opzionale.
- **Tipologia**: può assumere i seguenti valori:
  - F: funzionale;
  - Q: prestazionale;
  - P: qualitativo;

  - V: vincolo.

- **Codice**: numero progressivo identificativo strutturato nel formato: [codice_padre].[codice_figlio]

classificazione definisce l'importanza che, nonostante sia già presente nel codice identificativo, aumenta la facilità di lettura e di comprensione. **Descrizione** fornisce una spiegazione concisa, ma completa, del requisito. **Fonti** possono essere:

- capitolato$_G$: il requisito è stato quindi individuato dalla lettura del capitolato$_G$;
- interno: il requisito è stato individuato ed aggiunto in seguito ad un'analisi interna;
- caso d'uso$_G$: il requisito è stato individuato dallo studio di un caso d'uso$_G$;
- proponente: il requisito è stato individuato in seguito ad un colloquio con il proponente.

Diagrammi UML

I diagrammi UML$_G$ vengono realizzati usando la versione 2.0 del linguaggio con il servizio online LucidChart. Per la creazione si deve procedere in questo modo:

- selezionare il bottone "+ Document";
- realizzare il diagramma come normato in §2.2.4.1;
- andare nella sezione "My Documents", passare con il puntatore sopra al diagramma appena creato;
- selezionare il bottone "Share" e poi "Get shareable link";
- incollare in link dentro un apposito documento dentro la cartella Google Drive del gruppo;
- una volta verificati, i diagrammi sono esportati in formato png e inseriti nei documenti.

Per garantirne la leggibilità seguiamo le seguenti regole:

- gli elementi devono essere distribuiti in modo omogeneo e, se possibile, allineati sia in senso verticale che in senso orizzontale;

- mantenere uniforme la spaziatura tra gruppi analoghi di elementi della stessa tipologia;
- tutti gli attori principali sono raggruppati nella parte sinistra del diagramma mentre quelli secondari sono raggruppati nella parte destra.

2.2.4.1.1 Metriche di qualità

Le metriche di qualità utilizzate per valutare la qualità dell'analisi dei requisiti sono:

- **M01 Scostamento dei requisiti individuati**: indica la differenza in numero dei requisiti individuati in ogni periodo. Se elevato, questo valore indica che l'attività di analisi dei requisiti non è stata svolta con qualità sufficiente.

  formula: $R_{p-1} - Rp$ dove Rp indica il numero di requisiti individuati in un periodo e $Rp - 1$ indica il numero di requisiti individuati nel periodo precedente.

2.2.4.2 Progettazione

Scopo

Lo scopo della progettazione$_G$ è descrivere una soluzione soddisfacente per gli stakeholder$_G$. Questa attività definisce l'architettura logica del prodotto$_G$ software richiesto, a partire dall' analisi dei requisiti. Deve precedere la codifica e permette di:

- governare la complessità del prodotto organizzando e ripartendo i compiti implementativi;
- garantire la qualità, cioè l'efficacia del prodotto;
- garantire l'efficienza nella produzione attraverso l'ottimizzazione dell'uso delle risorse.

Viene quindi definita l'architettura del prodotto$_G$ finale che dovrà possedere le seguenti qualità:

- **Sufficienza**: deve essere in grado di soddisfare i requisiti individuati nel documento *Analisi dei Requisiti*;
- **Comprensibilità**: deve essere capita da tutti gli stakeholder$_G$;
- **Modularità**: deve essere suddivisa in parti chiare e distinte;
- **Semplicità**: ogni parte non deve contenere niente di superfluo;
- **Incapsulazione**: l'interno delle parti non deve essere visibile dall'esterno;
- **Coesione**: le parti che vengono unite devono avere gli stessi obiettivi;
- **Basso accoppiamento**: si devono evitare le dipendenze fra le parti;
- **Robustezza**: deve essere capace di sopportare ingressi diversi dall'utente e dall'ambiente;
- **Flessibilità**: deve permettere di attuare modifiche per fronteggiare i cambiamenti dei requisiti a costi contenuti;
- **Riusabilità**: possono essere impiegate parti in altre applicazioni;
- **Efficienza** nell'utilizzo delle risorse;
- **Affidabilità**: deve svolgere il compito per cui è stata ideata;

- **Disponibilità**: il sistema deve essere interrotto per un tempo limitato o non essere interrotto durante le operazioni di manutenzione;
- **Sicurezza** rispetto a malfunzionamenti e intrusioni.

Descrizione

La progettazione$_G$ è composta da due sezioni:

- **Technology baseline$_G$**: contiene le specifiche della progettazione ad alto livello del prodotto e delle sue componenti, i diagrammi UML che la descrivono e i test di verifica;
- **Product baseline$_G$**: integra ciò che è stato definito nella technology baseline$_G$ rendendo più dettagliata la progettazione; contiene inoltre i test di verifica.

I progettisti devono seguire le seguenti regole per progettare un'architettura di qualità:

- ove necessario, implementare i design pattern$_G$;
- evitare la creazione di package vuoti;
- evitare la creazione di classi e parametri non utilizzati;
- evitare la creazione di dipendenze circolari;
- ove possibile, prediligere l'utilizzo di interfacce e classi astratte;
- utilizzare nomi significativi per le classi e i metodi;
- assegnare le visibilità strettamente necessarie seguendo il principio di information hiding.

Technology Baseline

Nella technology baseline$_G$ sono presenti:

- **Tecnologie utilizzate**: descrizione di tutte le tecnologie che vengono utilizzate per l'implementazione del prodotto$_G$ con l'indicazione di vantaggi e svantaggi;
- **Proof of Concept (PoC)**: implementazione di un PoC$_G$ che utilizzi e dimostri la fattibilità delle tecnologie che utilizziamo per implementare il prodotto$_G$ software;
- **Tracciamento delle componenti**: riferimento di ogni requisito al componente che lo soddisfa;

Proof of Concept Abbiamo deciso di implementare un PoC_G in modo incrementale. Perciò, in accordo con il proponente, vengono selezionati degli incrementi da sviluppare. Questa decisione viene presa con l'intenzione di esplorare tutte le tecnologie che devono essere utilizzate nel prodotto_G finale. Queste tecnologie sono:

-
-

Product Baseline

Nella $\text{product baseline}_G$ sono presenti:

- **Design pattern**: viene fornita la descrizione dei design pattern utilizzati all'interno dell'architettura. Per ciascuno di essi deve essere presentato un diagramma che ne definisce la struttura e una descrizione;
- **Diagrammi UML**: per rendere più chiare e complete le scelte progettuali adottate, vengono utilizzati dei diagrammi UML usando la versione 2.0 del linguaggio. Ogni diagramma deve essere seguito dalla descrizione di ciò che rappresenta.

  Diagrammi UML Gli utilizzi dei diagrammi UML sono i seguenti:

  - **Diagrammi delle classi**: descrivono gli elementi del sistema e le loro interazioni;
  - **Descrizione delle classi**: descrizione degli obiettivi e delle funzionalità di ogni classe seguendo il seguente schema:
    - nome;
    - visibilità;
    - attributi;
    - metodi;
    - scopo e funzionalità.
  - **Tracciamento delle classi**: riferimento di ogni requisito alla classe che lo soddisfa;
  - **Diagrammi dei package**: descrivono le dipendenze tra gli oggetti raggruppati in un package;
  - **Diagrammi di attività**: descrivono la logica procedurale, utili per descrivere gli aspetti dinamici dei casi d'uso_G;

- **Diagrammi di sequenza**: descrivono la collaborazione di più elementi volti all'implementazione di un dato comportamento;
- **Test di sistema**: test necessari per verificare il corretto funzionamento dell'intero prodotto$_G$ software;
- **Test di integrazione**: test necessari per verificare che l'unione delle parti funzioni correttamente;
- **Test di unità**: necessari per verificare il corretto funzionamento di ogni singola parte del software.

Integrazione

L'attività di integrazione del software avviene in parte in automatico ed in parte tramite azioni e verifiche manuali.
Per l'integrazione automatica utilizziamo le Github Actions, offerte da Github, con cui implementiamo la continuous integration. L'integrazione continua automatica viene eseguita ad ogni push effettuato sul repository remoto ed esegue la compilazione, i test di unità, analisi statica$_G$ del codice per verificare che soddisfi i livelli di qualità desisderati e calcola il code coverage. I servizi utilizzati per eseguire i controlli sono JEst per i test d'unità, Coveralls per il code coverage e SonarCloud per l'analisi statica$_G$ del codice.
L'integrazione manuale consiste invece nel verificare il corretto funzionamento dell'applicativo esterno di addestramento, del plugin Grafana$_G$ e del loro dialogo. La verifica dei singoli componenti verrà automatizzata, per quanto possibile, tramite Selenium$_G$. Queste integrazioni vengono effettuate in seguito all'implementazione di ogni incremento del modello di sviluppo incrementale, quindi alla creazione di ogni nuova baseline$_G$.

2.2.4.3 Codifica

Scopo

Lo scopo di questa attività è dare una norma che definisca tutte le regole di codifica per garantire leggibilità e manutenibilità del codice. L'obiettivo della codifica è creare un prodotto$_G$ coerente con i requisiti e le aspettative del proponente, il cui codice sia leggibile, uniforme e permetta di eseguire più facilmente le attività di manutenzione e verifica. Dunque i programmatori sono tenuti a seguire tali regole durante l'attività di codifica.

Descrizione

Il prodotto$_G$ dell'attività di codifica è il codice necessario per realizzare il

capitolato$_G$. La sua struttura dovrà rispettare le regole definite nel documento *Piano di Qualifica* al fine di garantire una buona qualità.

Linguaggi di programmazione

I principali linguaggi di programmazione utilizzati sono:

- **JavaScript**: linguaggio di scripting attraverso il quale si intende sviluppare l'applicativo di addestramento esterno. Tramite la sua estensione JSX (JavaScript XML) è in grado di incorporare la sintassi HTML;
- **React**: libreria JavaScript usata per costruire elementi per interfaccia utente dell'applicativo esterno;
- **D3.js**: libreria JavaScript usata per visualizzare i grafici nell'applicativo esterno partendo da dei dati organizzati;
- **TypeScript**: estensione di JavaScript tramite la quale si intende sviluppare il plug-in su Grafana$_G$;
- **AngularJS**: framework JavaScript che permette di estendere il codice HTML attraverso l'utilizzo di direttive ed espressioni;
- **Plotly.js**: libreria JavaScript (basata su D3.js) che permette di visualizzare grafici personalizzati su Grafana$_G$;
- **HTML5**: linguaggio di mark-up che nell'ambito del progetto$_G$ viene principalmente generato dall'estensione JSX e gestisce il front-end dell'applicazione;
- **CSS3**: linguaggio usato per formattare in modo corretto documenti HTML5. Nell'ambito di questo progetto$_G$ viene utilizzato per impartire regole stilistiche al codice generato per gestire il front-end e la rappresentazione degli elementi grafici.

Tutti i linguaggi elencati devono rispettare le regole indicate nel seguente paragrafo ove applicabili.

Stile di codifica

In seguito vengono elencate le norme da rispettare nell'attività di codifica:

- **Nomenclatura**: i nomi di classi, metodi e variabili devono essere univoci, esplicativi ed uniformi. Segue la nomenclatura decisa per i vari elementi del codice:

- **Classi**: il nome di ciascuna classe deve seguire la pratica del PascalCase, cioè la prima lettera di ogni parola deve essere maiuscola. Segue un esempio corretto di dichiarazione di una classe:

```
// OK!
class MyClass {
        //...
}
```

- **Costanti**: i nomi delle costanti devono essere seguendo la pratica dell'ALL_CAPS, cioè tutte le parole in maiuscolo e divise dal carattere underscore (_). Segue un esempio corretto di dichiarazione di una costante:

```
const CONSTANT_VALUE = "This string won't change";
```

- **Metodi**: il nome di ciascun metodo deve iniziare con una lettera minuscola. Nei metodi con nome composto, la prima parola avrà la prima lettera minuscola, mentre le seguenti avranno la prima lettera maiuscola, come indicato dalla pratica del camelCase. Segue un esempio della corretta implementazione di suddetta pratica:

```
class MyClass {
// OK!
myMethod() {
        //...
}
}
```

- **Variabili**: il nome di ciascuna variabile deve iniziare con una lettera minuscola. Nelle varibili con nome composto, la prima parola avrà la prima lettera minuscola, mentre le seguenti avranno la prima lettera maiuscola, come indicato dalla pratica del camelCase. Segue un esempio della corretta implementazione di suddetta pratica:

```
// OK!
var myVar = 0;
```

- **Intestazione dei file**: ogni file deve avere un'intestazione contenuta in un blocco di commento dove vengono indicati:
  - nome completo del file con relativa estensione;

- autore del file;
- data di creazione;
- breve descrizione del contenuto del file.

```
/**
* File: fileName
* Author: Name Surname
* Creation date: YYYY-MM-DD
* Description: This header is for demonstration purpose only.
*/
```

- **Indentazione**: i blocchi annidati devono essere indentati con una tabulazione di quattro spazi. A tal fine gli IDE di tutti i componenti del gruppo dovranno essere configurati per rispettare suddette indicazioni. Segue un esempio di indentazione corretta:

```
// OK!
function() {
        var x = 1;
}
```

- **Commenti**: i commenti nel codice devono essere concisi ma sufficientemente descrittivi. Essi precedono l'implementazione di metodi e classi descrivendo brevemente la loro funzione. Sono possibili due tipi di commenti: in linea tramite l'utilizzo del doppio slash (//) o a blocco tramite la costruzione //**...*/. È inoltre possibile usare i commenti //TODO e //FIXME che indicano rispettivamente delle sezioni di codice da fare in un secondo momento e sezioni di codice da correggere. Segue un esempio di queste pratiche:

```
//** This is a test class.
* OK!
*/
class MyClass {
        // This is a test method.
        myMethod() {
                var myVar = undefined; //FIXME
                if (condition) {
                        //TODO
                } else {
                        //TODO
```

```
                }
        }
}
```

- **Blocchi if-else**: ogni blocco if deve avere un corrispondente blocco else che verrà posizionato in linea con la parentesi graffa di chiusura del blocco if precedente. Se i blocchi sono di una singola riga la parentesi graffa può essere omessa. Segue un esempio della corretta applicazione di questa pratiche:

```
// OK!
if (condition) {
        //TODO
} else {
        //TODO
}

if (condition)
return true;
```

- **Parentesizzazione**: le parentesi di delimitazione dei costrutti devono essere inserite in linea precedute da uno spazio. Segue esempio di utilizzo corretto di tale pratica:

```
// OK!
class MyClass {
        //TODO
}
```

- **Apici**: gli apici da utilizzare per la scrittura di stringhe sono i doppi apici ("). L'utilizzo dell'apice singolo è giustificato solo nel caso in cui la stringa presa in esame necessiti della presenza di apici al suo interno. Segue un esempio esplicativo:

```
// OK!
var myString1 = "This is a string";
var myString2 = "He said 'This is a string'";
```

- **Spaziatura tra operandi**: prima e dopo ciascun operando vanno inseriti degli spazi al fine di rendere più ordinato e leggibile il codice. Segue un esempio di tale pratica:

```
// OK!
var myVar1 = 1;
var myVar2 = 2;
var myVar3 = myVar1 + myVar2;
```

- **Dichiarazioni in linea**: è sconsigliata la dichiarazione di variabili in linea a favore dell'ordine e della leggibilità del codice. Segue un esempio esplicativo:

```
// NO!
var x = 1, y = 2;
// OK!
var x = 1;
var y = 2;
```

- **Spaziatura del codice**: tra i costrutti di codice è obbligatorio lasciare una riga vuota per rendere più ordinato e leggibile il codice. Segue un esempio di tale pratica:

```
// OK!
function() {
        //TODO
}

myMethod() {
        //TODO
}
```

- **Struttura dei metodi**: ogni metodo deve avere uno ed un solo compito e il suo contenuto dev'essere il più possibile breve in termini di righe di codice;
- **Condizioni**: l'uso di condizioni multiple è da evitare ove possibile per limitare la complessità delle singole espressioni;
- **Lingua**: la lingua con cui vengono scritti codice e commenti deve essere l'inglese;
- **Ricorsione**: l'uso della ricorsione è da evitare ove possibile in quanto aumenterebbe la complessità computazionale del codice.

Linee guida

Per quanto concerne i restanti aspetti riguardanti lo stile e le best practises sul codice vengono prese come riferimento dal gruppo le seguenti fonti:

- **SonarJS**: insieme di regole riguardanti JavaScript che, oltre alla segnalazione di bug, indicano anche delle linee guida per una migliore fruizione del codice. Tale obiettivo viene raggiunto anche grazie all'utilizzo di SonarLint, uno strumento atto all'analisi statica$_G$ del codice per segnalare eventuali incongruenze con le best practises `https://www.sonarsource.com/products/codeanalyzers/sonarjs.html`
- **SonarTS**: insieme di regole riguardanti TypeScript che, oltre alla segnalazione di bug, indicano anche delle linee guida per una migliore fruizione del codice. Tale obiettivo viene raggiunto anche grazie all'utilizzo di SonarLint, uno strumento atto all'analisi statica$_G$ del codice per segnalare eventuali incongruenze con le best practises `https://www.sonarsource.com/products/codeanalyzers/sonarts.html`
- **W3C**: per quanto riguarda l'HTML5 e il CSS3 il gruppo si attiene alle regole imposte dal World Wide Web Consortium. Per assicurarsi il raggiungimento di tali norme il codice prodotto verrà verificato tramite i validatori offerti da suddetto consorzio. Di seguito il link ai validatori utilizzati:
  - `https://validator.w3.org/`;
  - `https://jigsaw.w3.org/css-validator/`.

2.2.4.3.1 Metriche di qualità Per la codifica le metriche di qualità sono:

- **M02 Numero di parametri per metodo**: indica il numero di parametri presenti in ciascun metodo. Se il valore è elevato è molto probabile che il grado di complessità del metodo sia elevato. Perciò è opportuno limitare questo numero ed eventualmente suddividere il metodo in più metodi distinti;

  formula: si calcola con un valore intero.

- **M03 Numero di metodi per classe**: indica il numero di metodi presenti in ciascuna classe. Se il valore è elevato è molto probabile che il

grado di complessità del metodo sia elevato. Perciò è opportuno limitare questo numero in modo da avere una classe facilmente comprensibile e con uno scopo preciso;

formula: si calcola con un valore intero.

2.2.4.4 Integrazione software

2.2.5 Strumenti di supporto

Nella valutazione dei costi da affrontare per lo sviluppo del software e per la formazione del personale, abbiamo già identificato le principali tecnologie e strumenti che verranno utilizzare durante il progetto$_G$.

2.2.5.1 TeXstudio

Per la stesura dell'*Analisi dei Requisiti* viene utilizzato TeXStudio come ambiente di sviluppo.

2.2.5.2 WebStorm

Per la codifica viene utilizzato WebStorm, un ambiente di sviluppo integrato per JavaScript che offre piena compatibilità con Windows, Linux e SonarJS$_G$.

2.2.5.3 LucidChart

Per la realizzazione dei diagrammi UML$_G$.

3 Processi di supporto

3.1 Documentazione

3.1.1 Scopo

L'obiettivo di questo processo$_G$ è stabilire delle norme per la produzione, la verifica e l'approvazione finale della documentazione. Tali indicazioni devono essere condivise e seguite da tutti gli elementi del gruppo.

3.1.2 Aspettative

Questa sezione si prefigge due obiettivi:

- standardizzare la scrittura della documentazione al fine di offrire degli elaborati consistenti, coerenti tra di loro e modulari;
- dare delle norme per la produzione di documenti corretti e validi.

3.1.3 Descrizione

Questo processo descrive come il gruppo ha gestito il processo$_G$ di documentazione utile ai fini del progetto$_G$. Perciò si definiscono le fasi del ciclo di vita: costruzione del documento, strutturazione, stesura secondo gli standard tipografici utilizzati e gli strumenti, verifica e approvazione finale.

3.1.4 Attività

3.1.4.1 Ciclo di vita dei documenti

Ogni documento, durante il suo ciclo di vita, attraversa cinque fasi:

- **Creazione**: all'inizio il documento viene creato a partire da un template di base che abbiamo realizzato;
- **Struttura**: In seguito alla creazione, viene eseguita un attività di analisi del documento per delinearne la struttura principale. Essa è rappresentata dall'indice;
- **Stesura**: in questa fase viene effettuata la redazione del documento seguendo le norme indicate. Il responsabile, in seguito ad un'analisi preliminare, suddivide, se necessario, il lavoro tra vari analisti che procederanno alla stesura.

- **Verifica**: terminata la fase di stesura, i redattori sottometteranno l'elaborato al controllo dei verificatori. I verificatori, nominati in precedenza dal responsabile, analizzeranno il documento. Nel caso in cui si evidenzino delle criticità, segnaleranno le dovute modifiche ai redattori, in alternativa comunicheranno al responsabile la conformità del materiale.
- **Approvazione**: alla fine il documento viene vagliato dal responsabile che ne approva ufficialmente il rilascio.

3.1.4.2 Struttura dei documenti

3.1.4.2.1 Template LaTeX

Per la produzione di documenti consistenti e coerenti tra loro il gruppo ha deciso di creare un template LaTeX che permette, al momento della stesura, di concentrarsi unicamente sul contenuto e di non impegnare troppo tempo nella strutturazione della pagina. Il template è formato da più parti, tra cui:

- `frontmatter.tex`: file per la corretta formattazione della prima pagina, contiene anche dei comandi personalizzati per velocizzare l'adattamento del template alla rotazione dei ruoli;
- `package.tex`: file contenente i pacchetti LaTeX utilizzati per migliorare la rappresentazione dei contenuti;
- `config.tex`: file per la configurazione dei pacchetti inseriti in `package.tex`;
- `metadata.tex`: file per modificare i comandi personalizzati su `frontmatter.tex`, i quali garantiscono la modularità del template;
- `decision_table.tex`: file per la gestione del riepilogo tracciamenti relativo ai verbali;
- `log_table.tex`: file per la gestione del registro delle modifiche;
- `main.tex`: file, su cui si invoca la compilazione, che unisce le parti sopracitate con il documento vero e proprio.

3.1.4.2.2 Prima pagina

Il frontespizio è strutturato in questo modo:

- **Logo del gruppo**: corredato dal nome, il tutto centrato orizzontalmente;

- **Capitolato**$_G$: nome del progetto$_G$ scelto dal gruppo (*Predire in Grafana*), centrato orizzontalmente;
- **Titolo del documento**: nome esplicativo del contenuto del documento, centrato orizzontalmente e in grassetto;
- **Data**: data di approvazione finale dell'elaborato, centrata orizzontalmente;
- **Tabella informativa**: contenente informazioni quali:
  - versione del documento;
  - nominativo dell'approvatore (e quindi responsabile) del documento;
  - nominativo/i del/i redattore/i del documento;
  - nominativo/i del/i verificatore/i del documento;
  - stato del documento (approvato o non approvato);
  - tipo d'uso del documento (interno o esterno);
  - nominativi dei destinatari del documento;
  - email di riferimento del gruppo.
- **Descrizione**: breve riassunto del contenuto del documento.

3.1.4.2.3 Registro delle modifiche

La pagina successiva mostra una tabella contenente una raccolta cronologica delle modifiche avvenute sul documento. Le colonne espongono i seguenti contenuti:

- versione del documento dopo la modifica;
- data della modifica;
- nominativo del componente responsabile della modifica;
- ruolo del componente responsabile della modifica;
- breve descrizione della modifica effettuata.

3.1.4.2.4 Indice

L'indice riassume in modo schematico i contenuti del documento mostrando la loro suddivisione in sezioni, sottosezioni, paragrafi e sottoparagrafi. È presente all'inizio di ogni elaborato dopo il registro delle modifiche. I suoi elementi sono cliccabili e rimandano alla pagina corrispondente. In alcuni documenti viene corredato di due liste:

- **Lista delle immagini**: riportante le immagini utilizzate tramite rispettiva didascalia;
- **Lista delle tabelle**: riportante le tabelle utilizzate tramite rispettiva didascalia (sono esclusi il registro delle modifiche e il riepilogo tracciamenti).

3.1.4.2.5 Contenuto

Il corpo principale del documento è così strutturato:

- intestazione in alto con logo del gruppo riadattato a sinistra e nome del documento con relativa versione a destra;
- una riga divisoria;
- il contenuto della pagina formattato con una dimensione di 12 punti e la visualizzazione in A4 formato articolo (`\documentclass{article}`);
- una riga divisoria opzionale in caso di appunti;
- eventuali note sul testo riportate con un numero progressivo a esponente;
- una riga divisoria;
- il contenuto a piè di pagina con a destra il numero progressivo della pagina visualizzata sul totale.

3.1.4.2.6 Verbali

La struttura è schematica e fissa. Perciò risulta essere di più facile consultazione e uguale per tutti i verbali sia interni che esterni. Il documento è quindi composto dalle seguenti sezioni e sottosezioni:

- **Informazioni generali**
  - **Informazioni incontro**: contiene le informazioni riguardanti data, luogo, tempistiche e partecipanti all'incontro;

  - **Argomenti trattati**: contiene, in modo sintentico, un elenco numerato dei temi trattati durante l'incontro.

- **Verbale**

  - **Punto X**: spiega, in maniera più dettagliata, gli argomenti solo accennati nella sottosezione precedente. In questo caso X assumerà un valore numerico intero corrispondente al numero dell'elemento dell'elenco della sottosezione "Argomenti trattati" che si sta analizzando.

3.1.4.3 Norme tipografiche

3.1.4.3.1 Nomi dei file

Per organizzare i file in modo più ordinato e corretto il gruppo ha scelto di seguire la pratica dello "snake_case"$_G$. Tale prassi, nell'ambito di questo progetto$_G$, è applicata seguendo tre regole:

- i file con nome composto da più parole useranno il carattere underscore (_) come separatore;
- i nomi dei file saranno scritti interamente in minuscolo;
- tutti i documenti saranno corredati del numero di versione, tranne per i verbali che indicheranno la data dell'incontro;
- non vengono tralasciate preposizioni di alcun tipo dal nome del file.

3.1.4.3.2 Glossario

- Ogni parola da inserire nel *Glossario* è corredata da una "G" a pedice per tutte le sue occorrenze;
- sono escluse dalla regola precedente i titoli e il nome delle sezioni del documento.

3.1.4.3.3 Stili di testo

- **Grassetto**: il grassetto viene usato solamente per evidenziare le parole a cui si vuole dare risalto e per la prima parola di un elenco puntato se quest'ultimo è del tipo: **Termine**: definizione/spiegazione;

- **Corsivo**: il corsivo viene usato per evidenziare i nomi dei documenti oltre che per il nome dell'azienda proponente (*Zucchetti*) e del gruppo (*VRAM Software*);
- **Maiuscolo**: in aggiunta alle normali regole dell'italiano le lettere maiuscole verranno utilizzate anche per riportare i nomi dei documenti e di certi capitoli.

3.1.4.3.4 Elenchi puntati

Ogni voce dell'elenco puntato sarà preceduta da un "•" (punto elenco) per quanto riguarda il primo livello e da un "-" per il secondo. Alla fine di ogni elemento, invece, si aggiunge il carattere ";" a meno che non sia l'ultimo dove va messo il segno di punteggiatura ".". Se la voce dell'elenco è del tipo "Termine: spiegazione/definizione" il termine va in grassetto e con la prima lettera maiuscola, altrimenti vengono utilizzate le normali regole grammaticali della lingua italiana.

3.1.4.3.5 Convenzioni

Vengono usate tre convenzioni per la data, l'orario (i quali prendono come riferimento ISO 8601) e la valuta. Le date saranno espresse nella forma:

YYYY-MM-DD

dove:

- **YYYY**: indica l'anno;
- **MM**: indica il mese e deve sempre avere due cifre;
- **DD**: indica il giorno e deve sempre avere due cifre.

Gli orari saranno invece rappresentati nella forma:

HH:MM

dove:

- **HH**: indica le ore (in formato 24 ore) e deve sempre avere due cifre;
- **MM**: indica i minuti e deve sempre avere due cifre.

La valuta infine sarà scritta nella forma:

XXXX,YY

dove:

- **XXXX**: rappresenta la parte intera della somma;
- **YY**: rappresenta la parte decimale della somma.

3.1.4.3.6 Riferimenti

I riferimenti, nel nostro caso, possono essere informativi o normativi. I riferimenti informativi sono delle linee guida a cui non è necessario aderire rigorosamente, al contrario invece i riferimenti normativi devono essere rispettati. I riferimenti possono essere le slide del corso, i libri di cui è obbligatorio citare edizione e parti prese in esame ed in generale ogni contenuto online e non, che può dare un valore aggiunto alla realizzazione del progetto$_G$. Questo documento e le risorse relative al capitolato$_G$ scelto (*Predire in Grafana*) sono da considerarsi sempre riferimenti normativi.

3.1.4.3.7 Commenti

LaTeX dà la possibilità di inserire commenti all'interno del codice tramite il carattere "%". Tali commenti possono essere usati dai componenti del gruppo per appuntare delle indicazioni riguardanti i contenuti da aggiungere o modificare in un secondo momento. È obbligatorio che queste note siano sempre provviste del carattere sopracitato e che non vengano pubblicate.

3.1.4.4 Produzione

Il progetto$_G$ prevede la stesura di documenti che si dividono in interni ed esterni a seconda dei destinatari. Questi elaborati verranno poi valutati in quattro revisioni. Tali elementi progettuali sono elencati e spiegati di seguito con le rispettive sigle e la denominazione del file in questione adottata dal gruppo (solo nel caso in cui il documento sia già attivo).

3.1.4.4.1 Documenti esterni

- **Analisi dei Requisiti - AdR (analisi_dei_requisiti_vx.y.z)**: contiene la descrizione degli attori$_G$ del sistema e la loro interazione con i casi d'uso$_G$, fornendo una visione chiara ai progettisti sul problema da trattare;

- **Manuale Utente - MU**: serve per guidare l'utente nell'utilizzo del software;
- **Manuale Sviluppatore - MS**: ha lo scopo di spiegare a eventuali sviluppatori esterni il funzionamento del sistema;
- **Piano di Progetto - PdP (piano_di_progetto_vx.y.z)**: dà indicazioni sulle tempistiche e i costi che il gruppo prevede d'impiegare per la realizzazione del progetto$_G$;
- **Piano di Qualifica - PdQ (piano_di_qualifica_vx.y.z)**: fornisce le specifiche riguardanti il controllo qualità dei processi$_G$ e dei prodotti$_G$ tramite delle metriche$_G$ misurabili.

3.1.4.4.2 Documenti interni

- **Glossario - G (glossario_vx.y.z)**: documento dove sono riportate i termini che necessitano di spiegazione o che potrebbero causare confusione o fraintendimenti;
- **Norme di Progetto - NdP (norme_di_progetto_vx.y.z)**: linee guida per la gestione delle attività di progetto$_G$;
- **Studio di Fattibilità - SdF (studio_di_fattibilità_vx.y.z)**: breve analisi di tutti i capitolati$_G$ proposti.

3.1.4.4.3 Verbale

Il verbale della riunione è un documento particolare in quanto può essere sia interno che esterno. Esso è interno quando i partecipanti saranno solo i componenti del gruppo; esterno quando saranno presenti anche referenti dell'azienda. I file in questione sono denominati (seguendo le convenzioni riguardanti la nomenclatura e le date) come: "verbale_YYYY_MM_DD.tex". È presente, solo in questo tipo di documenti, la tabella "Riepilogo dei tracciamenti" che tiene segno delle modifiche discusse e approvate in sede d'incontro. Questo prospetto presenta una colonna con un codice nel formato:

V(I/E)_X.Y

dove:

- **V(I/E)**: indica se il verbale è interno o esterno;
- **X**: indica il numero dell'incontro che si sta organizzando;
- **Y**: indica il numero della scelta.

3.1.4.4.4 Revisioni

- **Revisione dei Requisiti - RR**: analisi iniziale del capitolato per concordare i requisiti col proponente;
- **Revisione di Progettazione - RP**: revisione intermedia per accertare la fattibilità del progetto$_G$;
- **Revisione di Qualifica - RQ**: revisione intermedia per l'approvazione delle verifiche e l'attivazione del processo$_G$ di validazione$_G$;
- **Revisione di Accettazione - RA**: revisione finale per il collaudo del software e l'accertamento del soddisfacimento dei requisiti presentati in RR.

3.1.4.4.5 Sigle dei ruoli

- **Responsabile - Re**;
- **Amministratore - Am**;
- **Analista - An**;
- **Progettista - Pr**;
- **Programmatore - Pt**;
- **Verificatore - Ve**;

3.1.4.5 Elementi grafici

3.1.4.5.1 Immagini

Le immagini sono sempre centrate e presentano una didascalia, tranne il logo nella prima pagina.

3.1.4.5.2 Tabelle

Le tabelle sono sempre centrate e presentano una didascalia, tranne il "Registro delle modifiche" e il "Riepilogo tracciamenti". Inoltre le tabelle presentano un'alternanza di colori per favorirne la lettura.

3.1.4.5.3 Didascalie

Le didascalie indicano in maniera progressiva il numero di tabella o immagine del documento e descrivono brevemente il loro contenuto.

3.1.5 Strumenti di supporto

3.1.5.1 LaTeX

Per stilare i documenti è stato usato il linguaggio di markup LaTeX. Questo linguaggio permette di scrivere dei testi mantenendo il focus sul contenuto e non sulla forma. La struttura infatti è stata dichiarata all'inizio del progetto$_G$ e poi riutilizzata permettendo una gestione degli elaborati scalabile, modulare e ordinata.

3.1.5.2 TeXStudio

Per scrivere in LaTeX viene usata l'IDE TeXStudio creata appositamente per questo fine e che quindi offre delle scorciatoie per i vari comandi, un correttore automatico per la lingua italiana e un servizio di compilazione e visualizzazione dei PDF integrato;

3.1.5.3 Lucidchart

Per la creazione dei diagrammi UML$_G$ e di altro tipo viene usata la piattaforma online Lucidchart che consente agli utenti di collaborare alla stesura, alla revisione e alla condivisione di grafici.

`https://www.lucidchart.com`

3.2 Gestione della configurazione

3.2.1 Scopo

L'obiettivo di questo processo$_G$ è rendere documenti e codice sorgente univocamente identificati e facilmente riconoscibili, evidenziando versioni e modifiche. Vuole anche agevolare l'identificazione delle relazioni esistenti fra gli elementi e fa da supporto alla fase di verifica.

3.2.2 Aspettative

Questa sezione ha lo scopo supportare i processi$_G$ di documentazione, di sviluppo e manutenzione del software rendendoli definiti e ripetibili.

3.2.3 Descrizione

Questo processo descrive come il gruppo ha gestito la configurazione degli strumenti e delle risorse utilizzate per svolgere il progetto$_G$. Sarà quindi descritta la configurazione del repository$_G$ su GitHub, del sistema di versionamento$_G$ Git e dei servizi GitHub.

3.2.4 Attività

3.2.4.1 Versionamento

Le modifiche effettuate ai documenti ed ai file contenenti codice sorgente sono identificate da un numero di versione presente all'interno dei file stessi. Per quanto riguarda i documenti, ogni numero di versione è presente nel nome del file ed ha una corrispondente riga nella tabella delle modifiche così da avere uno storico delle modifiche effettuate al documento. Per il codice sorgente del software le modifiche effettuate in una versione, sono consultabili sulla sezione rilasci di GitHub e saranno identificate da opportuni tag.
La numerazione per il versionamento del codice sorgente applica in modo indipendente alle singole componenti software che compongono il prodotto finale. Tuttavia esiste un'unica versione per il prodotto$_G$ finale, che segue le linee guida del versionamento del codice sorgente basate su risoluzione di problemi e nuove funzionalità minori/maggiori. Nelle note di rilascio del prodotto$_G$ finale, saranno poi riportate le versioni dei singoli componenti che lo compongono ovvero documentazione, plugin Grafana$_G$ e applicativo di addestramento esterno. Le modifche ed i rilasci saranno tracciabili tramite il repository$_G$ principale grafana_prediction che contiene al suo interno gli altri repository$_G$ come sottomoduli.

Numerazione delle versioni del documento Per identificare univocamente le versioni dei documenti seguiamo le indicazioni dell'ente ETSI, dunque la numerazione delle versioni è composta da 3 cifre nel formato X.Y.Z e la prima bozza ha versione 0.0.1. Le modifiche al documento aggiorneranno le cifre nel seguente modo:

- **Z** : Questa cifra viene incrementata ogni volta che vengono effettuate delle modifiche editoriali$_G$ al documento, es. X.Y.1, X.Y.2 etc.;
- **Y** : Questa cifra viene incrementata ogni volta che vengono effettuate delle modifiche tecniche$_G$ al documento. Se sono state eseguite modifiche sia editoriali$_G$ che tecniche$_G$ allora entrambe le cifre Z e Y saranno incrementate, es. X.1.1, X.2.2 etc;
- **X** : Questa cifra viene incrementata ogni volta che viene rilasciata una nuova versione finale del documento, ovvero una versione di rilascio.

La prima versione finale ha versione 1.1.1.
Durante la modifica e la revisione del documento le bozze vengono incrementate le cifre Z ed Y, es. 1.2.0, 1.2.1, 1.3.0, etc.
Quando il documento viene approvato come finale allora viene incrementata la cifra X, ad esempio la bozza 1.5.3 diventa la versione finale 2.1.1.
Ogni incremento sulle singole cifre è rigorosamente un +1, non possono essere saltati numeri.

Numerazione delle versioni del codice sorgente Per identificare univocamente le versioni del codice sorgente utilizziamo una numerazione basata su 3 cifre nel formato X.Y.Z, la prima bozza ha versione 0.0.0 e le modifiche al codice aggiorneranno le cifre della versione nel modo seguente:

- **Z** : Questa cifra viene incrementata ogni volta che vengono effettuate delle modifiche al codice sorgente al fine di risolvere problemi riscontrati nel software;
- **Y** : Questa cifra viene incrementata ogni volta che vengono effettuate delle modifiche al codice sorgente al fine di aggiungere nuove funzionalità di entità minore al software, la nuova versione deve inoltre presentare piena compatibilità con le versioni precedenti. Se sono state eseguite modifiche sia per risolvere problemi che per aggiungere nuove funzionalità minori allora entrambe le cifre Z e Y saranno incrementate, es. X.1.1, X.2.2 etc;

- **X** : Questa cifra viene incrementata ogni volta che vengono effettuate modifiche al codice al fine di aggiungere nuove funzionalità di entità maggiore al software, ad esempio novità che cambino in modo importante l'uso del software o le funzionalità che esso offre. Un aumento di versione di questo tipo può, se necessario, non essere retrocompatibile. Un aggiornamento di questa cifra comporta l'azzeramento delle cifre Y e Z, es 2.0.0, 3.0.0 etc;

La prima versione finale ha versione 1.0.0.
Ogni incremento sulle singole cifre è rigorosamente un +1, non possono essere saltati numeri.

3.2.4.2 Gestione delle modifiche

Al fine di monitorare e limitare le modifiche al ramo principale del repository$_G$, master, è utilizzato il meccanismo di pull request fornito da GitHub. Ogni membro del gruppo può creare branch secondari, secondo il workflow$_G$ feature branch, su cui effettuare modifiche, tuttavia per unirle al branch master è necessario aprire una pull request che dovrà essere revisionata dai verificatori tramite i servizi di revisione integrati in GitHub. Una volta revisionata positivamente è compito del responsabile del documento approvare la pull request ed effettuare quindi l'effettiva unione delle modifiche nel branch master.
In sintesi, per effettuare modifiche ai file sono previsti i seguenti passaggi:

- contattare il responsabile del file affinché autorizzi la modifica del file stesso;
- creare un branch secondario ed effettuare le modifiche al file;
- aprire una pull request per unire le modifiche al ramo master;
- i verificatori revisionano la pull request ed eventualmente richiedono aggiornamenti;
- completata la revisione il responsabile approva la pull request.

Il meccanismo sopra esposto si applica anche al codice sorgente. Inoltre in esso sono presenti anche dei sistemi di verifica automatica e di continuous integration che possono bloccare le pull request qualora le modifiche effettuate non rispettino i livelli di qualità desiderati o interrompano la compilazione o il superamento dei test automatici sul codice. In particolare, gli strumenti utilizzati a questo scopo sono Github Actions per implementare la continuous integration, JEst per l'esecuzione dei test automatici sul codice sorgente ed i servizi Coveralls e SonarCloud per controllare la qualità del codice sorgente.

3.2.4.3 Repository

Per tenere traccia di versioni e modifiche fatte a documenti e codice è utilizzato il sistema di versionamento$_G$ distribuito Git, che può essere utilizzato tramite riga di comando o utilità grafiche come GitHub Desktop o GitKraken. La struttura dei repository$_G$ utlizzati è la seguente:

```
grafana_prediction
├── VRAM_SW_DOC_2020
├── grafana_prediction_plugin
└── prediction_configuration
```

grafana_prediction è il repository$_G$ principale e gli altri sono suoi sottomoduli, configurati tramite la funzionalità Git submodules di Git. Il repository$_G$ principale, contenente i sottomoduli, è ospitato sul sito GitHub all'indirizzo:

`https://github.com/VRAM-Software/grafana_prediction`

Struttura del repository VRAM_SW_DOC_2020 Al fine di fornire una navigazione agevole e standardizzata, il contenuto del repository$_G$ è organizzato in modo gerarchico tramite directory secondo il seguente schema:

```
root
├── RR
│   ├── Esterni
│   └── Interni
├── RP
│   ├── Esterni
│   └── Interni
├── RQ
│   ├── Esterni
│   └── Interni
├── RA
│   ├── Esterni
│   └── Interni
├── Guide
└── Template
    ├── Images
    ├── config
    ├── img
    └── res
```

Nel dettaglio:

- **RR**: contiene i sorgenti LaTeX dei documenti relativi alla revisione dei requisiti;
- **RP**: contiene i sorgenti LaTeX dei documenti relativi alla revisione di progettazione$_G$;
- **RQ**: contiene i sorgenti LaTeX dei documenti relativi alla revisione di qualifica;
- **RA**: contiene i sorgenti LaTeX dei documenti relativi alla revisione di accettazione;
- **Guide**: contiene brevi indicazioni interne su come usare i comandi LaTeX usati nei documenti e su come riutilizzare il template per generare nuovi documenti;
- **Template**: contiene i sorgenti LaTeX usati per generare la base comune di tutti i documenti.

È inoltre disponibile una directory condivisa su Google Drive, con la stessa struttura gerarchica sopra esposta, che contiene tutti e soli i file PDF in versione finale. Lo scopo di questa directory condivisa è facilitare la consultazione e la condivisione dei file PDF stessi.

3.2.4.4 Tipologie di file non accettate

Tramite un apposito file .gitignore presente nella root directory della gerarchia vengono definiti i tipi di file non accettati all'interno del repository$_G$. Vengono esclusi tutti i file di compilazione, compilati o temporanei in quanto il repository$_G$ dovrebbe contenere solamente i seguenti formati di file:

- file sorgente LaTeX con estensione .tex;
- file immagine, preferibilmente in formato .png;
- file testuali in formato .md o .txt;
- tutti i file di codice sorgente come .js, .ts etc.

3.2.5 Strumenti di supporto

3.2.5.1 GitHub

Strumenti GitHub utilizzati

In aggiunta ai servizi già elencati, al fine di migliorare efficacia ed efficienza, vengono utilizzate le funzionalità di "Issue Tracking System", "Milestone" e "Project Board" integrate in GitHub. Ognuna di queste funzionalità viene usata solo da chi autorizzato, ad esempio rilasci di versioni, creazione e chiusura di milestone sono concesse solo al responsabile di progetto$_G$.
Vengono inoltre utilizzate le "Labels" offerte dall'Issue Tracking System di Github per definire le tipologie di problemi e le loro priorità.
Per il repository$_G$ contenente il codice sorgente del software sono utilizzate le GitHub Actions al fine di implementare la pratica della continuous integration che, a sua volta, utilizza strumenti quali JEst, SonarCloud e Coveralls per eseguire le verifiche automatiche sul codice sorgente.

3.2.5.2 SonarCloud I repository$_G$ del codice sorgente sono collegati al servizio SonarCloud ed è inserita la sua analisi statica$_G$ del codice sorgente nei controlli eseguiti dalla continuous integration. Le pagine sonarcloud dei repository sono disponibili al seguente indirizzo:

`https://sonarcloud.io/organizations/vram-software/projects`

3.2.5.3 Coveralls Poiché SonarCloud, nella versione disponibile al momento dello svolgimento del progetto$_G$, non supporta l'analisi del code coverage tramite Github Actions, utilizziamo il servizio offerto da Coveralls che abbiamo integrato nella continuous integration. Le pagine Coveralls dei repository sono disponibili ai seguenti indirizzi:

`https://coveralls.io/github/VRAM-Software/grafana_prediction_plugin`

`https://coveralls.io/github/VRAM-Software/prediction_configuration_utility`

3.3 Garanzia della qualità

3.3.1 Scopo

Lo scopo di questo processo$_G$ è fornire adeguate garanzie di qualità in merito ai processi implementati durante lo svolgimento dell'intero progetto$_G$.

3.3.2 Aspettative

Le aspettative del processo$_G$ di garanzia della qualità sono riassunte nei seguenti punti che rappresentano gli obiettivi che vogliono essere raggiunti. Si vuole ottenere:

- qualità per tutta la durata del ciclo di vita del software;
- garanzia di raggiungimento degli obiettivi di qualità prestabiliti;
- soddisfazione del cliente in merito al prodotto$_G$ sviluppato;
- oggettività nella ricerca della qualità in modo da ottenerla in tutti i processi$_G$ e in tutti i prodotti$_G$;
- qualità misurabile attraverso delle metriche$_G$.

3.3.3 Descrizione

Per garantire la qualità dello sviluppo di questo progetto$_G$ si è fatto riferimento allo standard ISO 9000:2015. Esso afferma che il sistema di qualità, cioè l'insieme dei metodi e regole per garantire la qualità di un prodotto$_G$, è costituito da tre passaggi:

- sviluppo;
- controllo;
- miglioramento continuo.

Questi tre passaggi sono ripetuti per ogni processo$_G$ e il loro scopo è raggiungere gli obiettivi di qualità prefissati per il prodotto$_G$ software, i documenti e i processi. Per assicurare che la qualità sia misurabile, fissiamo delle metriche associate alle attività che ci permettono di misurarla. I valori soglia e le misurazioni di tali metriche sono contenute nel documento: *Piano di Qualifica v. 6.1.1*. La misurazione di queste metriche deve essere fatta al termine di ogni singolo breve periodo indicato nella progettazione inserita nel documento *Piano di Progetto v. 6.1.1* in modo da monitorare e apportare miglioramenti al nostro way of working$_G$ costantemente.

3.3.4 Metriche di qualità di processo

Il parametro che abbiamo utilizzato è il seguente:

- **M04 Percentuale di metriche soddisfatte**: indica la percentuale di metriche$_G$ soddisfatte rispetto alla totalità di metriche$_G$ utilizzare all'interno del progetto$_G$;

  Formula: $\frac{numero\ di\ metriche\ soddisfatte}{numero\ di\ metriche\ totali}$.

3.3.5 Strumenti di supporto

Gli strumenti utilizzati per il processo$_G$ di garanzia della qualità sono le metriche di qualità.

3.4 Verifica

3.4.1 Scopo

Il processo$_G$ di verifica ha lo scopo di accertare che i risultati ottenuti da tutte le attività, attuate in un periodo preso in considerazione, raggiungano tutti i requisiti richiesti senza aver introdotto degli errori. La verifica deve essere svolta durante il ciclo di vita del software, più precisamente al passare di ogni baseline$_G$.

3.4.2 Aspettative

La verifica viene effettuata per raggiungere i seguenti obiettivi:

- cercare consistenza, completezza e correttezza nelle singole attività;
- ottenere supporto per una successiva validazione$_G$ del prodotto$_G$;
- avere a disposizione dei criteri e procedure, comprensibili e affidabili, da seguire.

3.4.3 Descrizione

Il processo$_G$ di verifica assicura che tutte le attività svolte della fase in esame, attraverso analisi e test successivamente definiti, siano conformi alle aspettative.

3.4.4 Attività

Le attività che si effettuano durante la verifica sono due tipi di analisi: l'analisi statica$_G$ e l'analisi dinamica.

3.4.4.1 Analisi statica

L'analisi statica$_G$ è lo studio della documentazione e del codice sorgente, essa non richiede l'esecuzione delle singole parti del sistema software che si sta sviluppando, quindi viene applicata ad ogni prodotto$_G$ di processo$_G$.

Verifica della documentazione La verifica della documentazione può essere effettuata manualmente in due modi:

- **Walkthrough**: metodo di lettura in cui tutti i componenti del gruppo analizzano tutti i documenti e codice realizzato, senza tralasciare nulla,

in modo da rilevare eventuali errori. Questa ricerca è ad ampio spettro, infatti viene effettuata anche se a priori non si può sapere se verranno trovate anomalie;

- **Inspection**: metodo di lettura o desk check per cercare errori specifici utilizzando liste di controllo o check list, la tabella successiva rappresenta gli errori che i nostri verificatori hanno identificato come essere i più frequenti.

Oggetto da controllare	**Correzione**
Formato data dei documenti	Tutte le date devono utilizzare la seguente scrittura: *YYYY-MM-DD*
Sintassi nome documenti citati	I documenti, interni o esterni, citati nella documentazione devono essere scritti in italico utilizzando il comando: `\textit{}`
Sintassi parole da enfatizzare	Le parole da enfatizzare, come titolo di liste puntate, devono essere scritte i grassetto utilizzando il comando: `\textbf{}`
Punteggiatura liste puntate	Ogni elemento nelle liste puntate deve iniziare con una lettera maiuscola o minuscola (a seconda del contesto) e deve finire con un punto e virgola
Uso corretto dei comandi `\glo` e `\glosp`	I comandi `\glo` e `\glosp`, sono stati creati per identificare le parole da inserire nel glossario, `\glo` dovrà essere usato se la parola del glossario precede un carattere di punteggiatura. Il comando `\glosp` sarà invece usato per le parole che precedono uno spazio

Tabella 1: Tabella errori più frequenti

3.4.4.2 Analisi dinamica

L'analisi dinamica è una tecnica che consiste nell'esecuzione del prodotto$_G$ software eseguendo dei test su un insieme finito di casi.

3.4.4.2.1 Test

Per la corretta verifica del codice, i test devono seguire regole ben precise che definiscono i parametri e le proprietà che devono avere test per essere considerati efficaci:

Parametri da definire:

- **Input**: l'input preso in considerazione;
- **Output**: l'output che si aspetta di ricevere;
- **Ambiente**: l'ambiente, quindi hardware e sistema operativo, in cui viene eseguito il test;
- **Stato iniziale**: lo stato iniziale del prodotto$_G$, precedente all'esecuzione del test;
- **Istruzioni opzionali**: l'utilizzo di istruzioni opzionali aggiuntive deve essere noto.

Un test può essere definito efficace se:

- i parametri citati sopra sono correttamente indicati;
- il nome di un test deve essere autoesplicativo per capire subito che cosa si sta testando;
- esso è ripetibile, quindi ci si aspetta un risultato costante per tutte le esecuzioni di quel test;
- in caso di fallimento del test, esso devo fornire informazioni utili a risolvere il problema identificato;
- in caso di possibili effetti indesiderati ci si aspetta un avvertimento;
- esso è esaustivo e accurato, infatti devono poter identificare una parte del progetto$_G$ che potrebbe essere causa di errori.

Il software testing è un procedimento importante per misurare la qualità di un prodotto$_G$ software, per questo motivo sono definiti quattro livelli che dividono i tipi di test da effettuare secondo una gerarchia. Essi sono:

- **Test di accettazione**;
- **Test di sistema**;
- **Test di integrazione**;
- **Test di unità**.

Codifica dei test Per ogni test vengono definiti il codice identificativo, la descrizione, lo stato e l'esito.

- il codice identificativo è scritto in questo formato: **tipologia[codice_padre].[codice_figlio]**
  - **tipologia**:
    - **TA**: test di accettazione;
    - **TS**: test di sistema;
    - **TI**: test di integrazione;
    - **TU**: test di unità.
  - **codice_padre**: numero che identifica univocamente il test;
  - **codice_figlio**: numero progressivo che indica i sotto-test che compongono il padre.
- descrizione: spiegazione concisa e completa di ciò che verifica il test;
- stato:
  - **I**: implementato;
  - **NI**: non implementato.
- Esito:
  - **P**: positivo;
  - **N**: negativo;
  - **NE**: non eseguito.

Test di unità

I test di unità si definiscono tali perché controllano il corretto funzionamento di una sezione di codice specifica chiamata unità. Solitamente questi tipi di test vengono eseguiti su funzioni o, se si parla di programmazione ad oggetti, su un metodo di una classe, quindi su una parte minimale del codice. Il loro scopo è verificare che gli input forniti diano gli output attesi. È buona norma rendere questi test automatici per minimizzare il tempo di esecuzione senza richiedere l'interazione dello sviluppatore. Questo tipo di test viene scritto dal programmatore che sviluppa le singole unità mediante l'ausilio di driver$_G$ e stub$_G$, per verificare l'assenza di errori e documentare il funzionamento dell'unità.

Test di integrazione

Successivamente ai test di unità, si assemblano gruppi di unità progressivamente che formeranno degli agglomerati sempre più grandi. Questo tipo di test, infatti, si concentra nella ricerca di anomalie ed errori tra le interfacce dei singoli componenti. Una volta concluso un singolo test, si otterrà un agglomerato che costituirà una nuova unità da cui partire per ulteriori test di integrazione. Per svolgere questo test si è scelto un approccio iterativo.

Test di sistema

I test di sistema controllano il funzionamento corretto del prodotto$_G$ e verificano che il sistema soddisfi tutti i requisiti definiti nel documento *Analisi dei Requisiti v. 3.1.1*. Con l'esecuzione di questi test si riesce a definire e documentare tutte le funzionalità del sistema e possono individuare criticità come il problema del black-box$_G$. Il gruppo svolgerà questo test quando si raggiungerà uno stato di rilascio per il prodotto$_G$ sviluppato. Per identificare un test di unità viene utilizzata una codifica con il seguente formato:
TS[codice] Dove codice è un numero progressivo nel formato: [X].[Y] con X e Y numeri maggiori di zero.

Test di accettazione

I test di accettazione, o di collaudo, vengono effettuati ad ogni pre-rilascio, quindi dopo l'esecuzione dei test di sistema. Essi servono a confermare il soddisfacimento dei requisiti da parte del committente; infatti questi tipi di test ne richiedono la presenza. Se questi test vengono superati significa che il prodotto$_G$ software è pronto per il rilascio. Per identificare un test di accettazione viene utilizzata una codifica con il seguente formato:

TA[codice] Dove codice è un numero progressivo nel formato: [X].[Y] con X e Y numeri maggiori di zero.

3.4.4.2.2 Metriche di qualità Le metriche di qualità utilizzate per valutare la qualità della verifica sono:

- **M05 Percentuale bug sistemati**: indica la percentuale dei bug sistemati dando così una traccia dell'avanzamento;

  formula: $\frac{bug\ sistemati}{bug\ totali}$;

3.4.5 Strumenti di supporto

3.4.5.1 Correzione ortografica

Il nostro gruppo ha utilizzato, per la ricerca di errori ortografici o stilistici nel testo dei documenti, la libreria HunSpell utilizzata con l'editor di test: TexStudio. Esso permette di avere un controllo in tempo reale su errori ortografici in lingua inglese e italiana (sottolineati in rosso) e ripetizioni di parole a breve distanza (sottolineati in verde).

3.4.5.2 SonarJS

Per l'analisi statica$_G$ del codice Javascript viene utilizzato il code analyzer SonarJS$_G$ su piattaforma di SonarCloud.

3.4.5.3 SonarTS

Per l'analisi statica$_G$ del codice Typescript viene utilizzato il code analyzer SonarJS$_G$ su piattaforma di SonarCloud.

3.4.5.4 Validazione W3C Le pagine di markup scritte il linguaggio HTML5 vengono validate attraverso il validatore W3C reperibile al seguente indirizzo:
`https://validator.w3.org/` Le pagine di presentazione scritte in linguaggio CSS3 vengono validate attraverso il validatore W3C reperibile al seguente indirizzo:
`https://jigsaw.w3.org/css-validator/`

3.4.5.5 Jest Per la gestione dei test del codice JavaScript il nostro gruppo ha utilizzato il framework Jest per eseguire test d'unità sul codice React e NodeJS.

3.4.5.6 React Testing Library A supporto del framework Jest, viene utilizzata la libreria Testing Library per React che facilita la scrittura di test manutenibili per componenti React.

3.4.5.7 Coveralls Coveralls è uno strumento che permette di quantificare la percentuale di codice testato.

3.5 Validazione

3.5.1 Scopo

Lo scopo del processo$_G$ di validazione$_G$ è accertare che il prodotto$_G$, per uno specifico uso, corrisponda alle attese.

3.5.2 Aspettative

Le aspettative del gruppo sono di normare adeguatamente il processo$_G$ di validazione$_G$ al fine di rilasciare un prodotto$_G$ privo di errori gravi e capace di rispondere alle richieste del proponente.

3.5.3 Descrizione

Le attese sono quelle del committente e corrispondono, se stilata correttamente, all'*Analisi dei Requisiti v. 3.1.1*. Per eseguire una validazione$_G$ si deve avere un prodotto$_G$ che è a un sufficiente livello di avanzamento, quindi il numero di validazioni svolte è molto minore rispetto al numero delle verifiche, che iniziano a venire eseguite prima.

3.5.4 Attività

3.5.4.1 Validazione Ci sono due tipologie di validazione$_G$:

- interna: ha successo se i requisiti in precedenza scritti sono soddisfatti;
- esterna: ha successo se il committente fornisce un riscontro positivo ed è soddisfatto.

Più nel particolare la validazione$_G$ consiste nel:

- selezionare i requisiti da soddisfare;
- individuare dei test specifici per misurare il soddisfacimento dei requisiti selezionati;
- eseguire i test;
- analizzare i risultati dei test.

3.6 Gestione dei cambiamenti

3.6.1 Scopo

Lo scopo di questo processo$_G$ è fornire un metodo tempestivo, disciplinato e documentato per assicurare che tutti i cambiamenti, compresi i problemi, riscontrati nel corso del progetto$_G$, siano identificati, analizzati e risolti.

3.6.2 Aspettative

L'applicazione di questo processo$_G$ deve comportare maggior tempestività ed efficacia nell'individuazione e nella risoluzione dei problemi, con un conseguente aumento di efficacia ed efficienza del processo$_G$ di verifica. Inoltre questo processo$_G$ deve permettere di individuare i problemi maggiormente ricorrenti.

3.6.3 Descrizione

Il processo$_G$ di gestione dei cambiamenti analizza e risolve tutti i cambiamenti programmati e improvvisi, quali i problemi, riscontrati durante l'esecuzione dei processi$_G$ di sviluppo e di fornitura, indipendentemente dalla loro natura e dalla loro origine.

3.6.4 Attività

3.6.4.1 Implementazione del processo

Deve essere istanziato il processo$_G$ di gestione dei cambiamenti per gestire ogni nuova modifica, comprese le non conformità ed i problemi non pianificati, rilevato nei prodotti$_G$ software o nelle attività. Questo processo$_G$ deve essere conforme alle seguenti proprietà:

- il processo$_G$ deve essere circolare e chiuso, assicurando che:
  - tutti i cambiamenti siano prontamente segnalati e gestiti tramite il processo di risoluzione dei cambiamenti;
  - i cambiamenti vengano presi in carico e gestiti;
  - vengano inviate delle notifiche per informare gli interessati della presenza dell'eventuale problema;
  - le cause del problema vengano identificate, analizzate e, dove possibile, eliminate;
  - la risoluzione e le decisioni prese siano archiviate e storicizzate;

  - lo stato del cambiamento sia tracciato, aggiornato e comporti delle notifiche al cambio di stato;
  - venga mantenuto un registro di tutti i problemi riscontrati.

- Il processo$_G$ definisce uno schema per categorizzare e dare priorità ai cambiamenti. Questo schema è composto da:
  - **Identificativo**: codice numerico progressivo per identificare il singolo cambiamento;
  - **Tipologia**: denota il tipo di cambiamento e può essere:
    * **Editoriale**: cambiamento che comporta modifiche editoriali$_G$ sulla documentazione;
    * **Tecnico**: cambiamento che comporta modifiche tecniche$_G$ sulla documentazione;
    * **Funzionale**: cambiamento che influenza le caratteristiche funzionali del prodotto$_G$ software;
    * **Conformità**: cambiamento che comporta violazioni di conformità del prodotto$_G$ software;
    * **Validazione**: cambiamento riscontrato durante il processo$_G$ di Validazione$_G$ del prodotto$_G$ software.
  - **Priorità**: può essere:
    * bloccante;
    * urgente;
    * alta;
    * media;
    * bassa.
  - **Stato**: può essere:
    * da fare;
    * in corso;
    * completato.

- I cambiamenti rilevati ed archiviati vengono analizzati per identificare quelli più frequenti ed eventuali tendenze;

- La modalità di gestione e di risoluzione individuate per i cambiamenti e per i problemi sono analizzate e valutate al fine di verificare che siano stati effettivamente risolti e che le modifiche siano state correttamente implementate nei prodotti$_G$ software e nelle attività. Inoltre l'analisi deve permettere di individuare dei pattern di risoluzione ed aiutare a determinare quando vengono introdotti ulteriori problemi.

3.6.4.2 Risoluzione dei problemi

Quando dei cambiamenti, comprese le non conformità ed i problemi, sono rilevati in un prodotto$_G$ software o in un'attività, deve essere realizzato un report che li descriva. Questo report deve essere usato come componente del ciclo chiuso del punto §3.6.4.1: dall'individuazione dei cambiamenti al processo di indagine, analisi e risoluzione, fino all'individuazione della causa ed alla sua analisi per individuare le tendenze. Per svolgere quanto descritto, ci si appoggia all'Issue Tracking System di GitHub. Esso consente di aggiungere delle issues che corrispondono ai cambiamenti o ai problemi che si devono svolgere e di assegnarle ad incaricati e responsabili. Dunque viene assegnata un'istanza dello schema precedentemente descritto e, in base alle priorità, si procederà alla loro risoluzione.

3.6.4.2.1 Metriche di qualità La metrica di qualità utilizzata per valutare la qualità della risoluzione dei problemi è:

- **M06 Tempo medio di risoluzione degli errori**: indica il tempo medio di risoluzione degli errori segnalati. Se elevato, questo valore indica che la gestione dei cambiamenti non è adeguata;

  formula: $\frac{TRE}{NE}$ dove TRE indica il tempo totale per la risoluzione degli errori e NE il numero di errori risolti.

4 Processi organizzativi

4.1 Gestione organizzativa

4.1.1 Scopo

Lo scopo del processo$_G$ di gestione organizzativa è definito dai seguenti punti:

- creazione di un modello organizzativo che specifica i rischi che possono verificarsi;
- definizione un modello di sviluppo da seguire;
- pianificazione del lavoro in base alle scadenze fissate;
- creazione e calcolo di un piano economico suddiviso tra i ruoli;
- definizione finale del bilancio sul totale delle spese.

Queste attività sono definite a cura del responsabile di progetto$_G$ e redatte all'interno del documento *Piano di Progetto v. 6.1.1*.

4.1.2 Aspettative

Gli obiettivi del processo$_G$ di gestione organizzativa sono:

- coordinare i componenti del gruppo attraverso l'assegnazione di ruoli e compiti;
- coordinare la comunicazione tra i membri del gruppo con l'ausilio di strumenti che la rendano facile ed efficace;
- pianificare l'esecuzione delle attività in modo ragionevole;
- gestire le attività anche dal punto di vista economico;
- monitorare costantemente il team, i processi$_G$ e i prodotti$_G$ in modo efficace.

4.1.3 Descrizione

Il processo$_G$ di gestione organizzativa è composto dalle seguenti attività:

- definizione dello scopo;
- generazione delle istanze dei processi$_G$;

- stima e pianificazione di risorse, costi e tempo per lo svolgimento del progetto$_G$;
- assegnazione dei ruoli e, per ognuno di essi, delle attività;
- gestione del controllo e dell'esecuzione di tutte le attività;
- valutazione periodica dello stato e dell'esecuzione delle attività rispetto a quanto pianificato.

4.1.4 Metriche di qualità del processo

Pianificazione delle risorse

I parametri utilizzati per valutare la qualità della pianificazione sono:

- **M07 Planned value**: indica il valore dei lavoro pianificato fino a quel momento;

  formula: BAC · %lavoro pianificato con BAC=budget totale;

- **M08 Earned value**: indica il valore del lavoro svolto fino a quel momento;

  formula: BAC · %lavoro svolto BAC=budget totale;

- **M09 Actual cost**: indica il denaro speso fino a quel momento;

  formula: si calcola con un valore intero;

- **M10 Cost performance index**: rappresenta un indice per i costi effettivi rispetto a quelli previsti, se è $\leq$1 indica che si sta spendendo più di quanto preventivato;

  formula: $\frac{EV}{AC}$;

- **M11 Schedule performance index**: rappresenta un indice per i tempi effettivi rispetto a quelli previsti, se è $\leq$1 indica che si sta impiegando più tempo di quanto preventivato;

  formula: $\frac{EV}{PV}$;

- **M12 Estimated cost at completion**: rappresenta il budget totale stimato del progetto$_G$ con il CPI del momento;

  formula: $AC + \frac{BT-EV}{CPI}$;

- **M13 Schedule at completion**: rappresenta il tempo totale stimato del progetto$_G$ con SPI del momento;

  formula: $\frac{TT}{SPI}$; con TT=Tempo Totale.

4.1.5 Attività

4.1.5.1 Ruoli di progetto

4.1.5.1.1 Assegnazione

I ruoli scelti corrispondono alla rispettiva figura aziendale e valgono per la durata di una milestone. Al termine di ognuna di esse, viene eseguita una rotazione dei ruoli così da permettere a ciascuno dei membri del gruppo di ricoprirli tutti almeno una volta. I ruoli previsti sono descritti di seguito.

4.1.5.1.2 Responsabile di progetto

La figura del responsabile di progetto$_G$ è molto importante in quanto convergono su di esso le responsabilità di gestione, pianificazione, coordinamento e controllo dell'intero progetto$_G$. Le sue mansioni sono:

- controllo e coordinamento di risorse, componenti e attività del gruppo;
- relazione con il controllo di qualità interno al progetto$_G$;
- analisi e gestione degli elementi più critici e i rischi;
- intermediazione nelle comunicazioni esterne ovvero con proponente e committente;
- elaborazione ed emissione di piani e scadenza;
- approvazione finale dei documenti.

4.1.5.1.3 Amministratore di progetto

La figura dell'amministratore è responsabile dell'efficienza e dell'operatività dell'ambiente di sviluppo e della redazione e attuazione di piani e procedure di gestione per la qualità. Le sue mansioni sono:

- controllo versioni e configurazioni del prodotto$_G$;
- gestione di tutta la documentazione del progetto$_G$;
- direzione delle infrastrutture di supporto;

- risoluzione dei problemi sorti dalla gestione dei processi$_G$;
- stesura del documento *Norme di Progetto* per conto del responsabile di progetto$_G$;
- collaborazione con il responsabile di progetto$_G$alla reazione del documento *Piano di Progetto*.

4.1.5.1.4 Analista

La figura dell'analista è responsabile di tutta l'attività di analisi dei problemi e del dominio applicativo. Le sue mansioni sono:

- studio del dominio del problema;
- analisi della complessità, della fattibilità e dei requisiti del problema;
- stesura del documento *Studio di Fattibilità*;
- stesura del documento *Analisi dei Requisiti*.

4.1.5.1.5 Progettista

La figura del progettista è responsabile delle attività di progettazione$_G$. Deve perciò gestire gli aspetti tecnologici e tecnici del progetto$_G$. Le sue mansioni sono:

- redigere le specifiche tecniche del prodotto$_G$ effettuando scelte il più efficienti e performanti possibili, sulla base delle tecnologie presenti;
- sviluppare l'architettura del prodotto$_G$ tale da fornire una base stabile a manutenibile che soddisfi le specifiche tecniche.

4.1.5.1.6 Programmatore

La figura del programmatore è responsabile dell'attività di codifica mirata alla realizzazione del prodotto$_G$ compreso delle componenti necessarie allo svolgimento di test di verifica e validazione$_G$. Le sue mansioni sono:

- eseguire l'attività di codifica sulla base del lavoro svolto da progettista;
- fornire dei componenti che permettano si svolgere test e validazione$_G$ sul prodotto$_G$.

4.1.5.1.7 Verificatore

La figura del verificatore è responsabile di tutta l'attività di verifica. Si affida al documento *Norme di Progetto* è definito anche lo standard da seguire per questa attività. Le sue mansioni sono:

- redazione della parte retrospettiva del *Piano di Qualifica*;
- ispezione dei documenti al fine di controllare che siano conformi allo standard previsto nelle *Norme di Progetto*;
- segnalazione di eventuali errori nelle parti di prodotto$_G$ a chi ha responsabilità su di esse.

4.1.5.2 Gestione dei rischi

La gestione e l'analisi dei rischi è un'attività a carico del responsabile di progetto$_G$ che deve essere documentata nel documento *Piano di Progetto v. 6.1.1*. La procedura per la gestione dei rischi è composta dai seguenti passi:

- individuazione dei rischi;
- analisi dei rischi;
- pianificazione dell'attività di controllo;
- controllo e monitoraggio.

Viene definita una codifica per l'identificazione dei fattori di rischio: R[tipo][X]

- tipo:
  - **RT**: rischi tecnologici di lavoro e produzione del software;
  - **RG**: rischi di gruppo;
  - **RO**: rischi organizzativi delle attività da svolgere;
  - **RR**: rischi legati ai requisiti e ai rapporti con gli stakeholder$_G$;
  - **RS**: rischi di stima di costi e tempi.
- X: numero intero progressivo che inizia da 1.

Inoltre vengono definiti delle misure di probabilità e degli indici di gravità:

- probabilità:
  - **Alta**;

  - **Media**;
  - **Bassa**.

- Gravità:
  - **1**: basso livello di gravità;
  - **2**: medio livello di gravità;
  - **3**: alto livello di gravità.

4.1.5.2.1 Metriche di qualità I parametri utilizzati per valutare la qualità della gestione dei rischi sono:

- **M14 Rischi non preventivati**: indica il numero di rischi non preventivati che si verificano. Un valore elevato indica una superficialità nell'individuazione dei rischi possibili;

  formula: si calcola con un valore intero partendo da 0 e andando a fare un incremento ogni volta che si verifica un imprevisto.

4.1.6 Procedure

Sono state definite un insieme di procedure che definiscono la gestione del coordinamento interno del gruppo.

4.1.6.1 Gestione delle comunicazioni

4.1.6.1.1 Comunicazioni interne

Le comunicazioni interne al gruppo avvengono tramite la piattaforma Slack nella quale è stato creato un workspace suddiviso nei seguenti canali tematici:

- **Documentazione**: canale dedicato alle discussioni in merito alle attività di stesura e verifica della documentazione;
- **Sviluppo**: canale dedicato alle discussioni in merito alle attività di analisi, progettazione$_G$ e codifica del prodotto$_G$;
- **Generale**: canale dedicato alle comunicazioni di servizio per il coordinamento di attività, incontri e impegni del gruppo.

Nei canali dedicati alla documentazione e allo sviluppo è stata attivata l'integrazione con GitHub per segnalare i nuovi commit e le pull request con un messaggio. Inoltre è stata eseguita l'integrazione con Google Calendar che

permette di avere una sezione dedicata alle notifiche degli eventi programmati sul calendario e l'integrazione del plug-in Simple Poll che permette di eseguire dei sondaggi per prendere le decisioni.

4.1.6.1.2 Comunicazioni esterne

Le comunicazioni esterne avvengono tramite la posta elettronica all'indirizzo `vram.software@gmail.com`. Il responsabile di progetto$_G$ ha l'incarico di gestire queste comunicazioni ed è tenuto ad informare tutti gli elementi del gruppo degli argomenti trattati qualora non fossero presenti.

4.1.6.2 Gestione degli incontri

4.1.6.2.1 Incontri interni

Gli incontri del gruppo sono accordati su Slack nel canale generale oppure nella precedente riunione e vengono annotati su Google Calendar. Essi sono identificati da:

- data;
- ora di inizio;
- durata prevista;
- elenco di attività da svolgere.

4.1.6.2.2 Verbali di incontri interni

Un segretario, nominato dal responsabile, è incaricato della redazione di un *Verbale* della riunione che tiene traccia di:

- **Luogo**;
- **Data**;
- **Ora d'inizio**;
- **Ora di fine**;
- **Partecipanti**;
- **Argomenti trattati**: argomenti all'ordine del giorno che sono trattati nell'incontro da verbalizzare;
- **Discussioni e decisioni prese**.

4.1.6.2.3 Incontri esterni

Gli incontri esterni vengono concordati se i membri del gruppo, il committente o il proponente lo ritengono necessario. In tal caso, sarà compito del responsabile di progetto$_G$ definire data e ora dell'incontro in accordo tra le due parti attraverso gli appositi canali di comunicazione descritti precedentemente.

4.1.6.2.4 Verbali di incontri esterni

Un segretario, nominato dal responsabile, è incaricato della redazione di un *Verbale* della riunione tenendo traccia di:

- **Luogo**;
- **Data**;
- **Ora d'inizio**;
- **Ora di fine**;
- **Partecipanti**;
- **Argomenti trattati**: argomenti all'ordine del giorno che sono trattati nell'incontro da verbalizzare;
- **Verbale**: esposizione delle discussioni e delle decisioni prese.

4.1.6.3 Gestione degli strumenti di coordinamento

4.1.6.3.1 Ticketing

Il sistema di ticketing permette ai membri del gruppo di gestire le tutte le attività del progetto$_G$. Per eseguire il ticketing vengono utilizzati gli strumenti dell'Issue Tracking System presenti gratuitamente in GitHub ed il funzionamento è il seguente: il responsabile di progetto$_G$ ha l'incarico di assegnare i compiti ai membri del gruppo attraverso le issue. Ognuna di esse contiene titolo, descrizione, milestone, le project board Kanban automatizzate associate e il destinatario. Le project board Kanban automatizzate permettono al responsabile di progetto$_G$ di monitorare l'andamento delle attività tracciando lo stato di ogni singolo compito che può essere:

- To Do (da fare);
- In Progress (in lavorazione);

- Done (concluso).

L'associazione con le milestone$_G$ definisce le scadenze delle singole issue. Inoltre è presente una comoda funzionalità di ricerca per le issue assegnate che ne agevola ulteriormente la gestione.

4.1.7 Strumenti di supporto

Gli strumenti utilizzati sono i seguenti:

- **Telegram**: strumento di messaggistica istantanea utilizzato inizialmente dal gruppo per il primo incontro;
- **Slack**: strumento utilizzato per le comunicazione interne del gruppo;
- **Google Drive**: strumento utilizzato per file che non sono oggetto di cambiamenti frequenti come il *Glossario* o immagini di vario tipo;
- **Google Calendar**: strumento utilizzato per tracciare e semplificare la gestione degli impegni e fornire a tutti notifiche e promemoria per gli stessi;
- **Gmail**: strumento di posta elettronica utilizzato per le comunicazioni esterne.
- **Skype**: strumento utilizzato per effettuare chiamate VoIP per alcuni incontri interni;
- **Git**: sistema di controllo di versionamento$_G$;
- **GitKraken, GitHub Desktop**: interfacce l'utilizzo di GitHub in locale;
- **GitHub**: strumento utilizzato per il proprio Issue Tracking System e sistema di versionamento$_G$ dei file.

4.2 Formazione

4.2.1 Scopo

Lo scopo di questo processo$_G$ è formare ogni membro del gruppo per garantire che ognuno abbia le conoscenze e le competenze adeguate allo svolgimento dei compiti assegnati.

4.2.2 Aspettative

L'aspettativa del gruppo, nel caso di questo processo$_G$, è riuscire a ottimizzare i tempi non impiegando tutti i membri nell'apprendimento di uno stesso strumento o tecnologia, ma lavorare in modo parallelo, al fine di assimilare più concetti possibili per poi condividerli con gli altri componenti.

4.2.3 Descrizione

Il processo$_G$ di formazione è composto da un insieme di attività che definiscono come vengono formati i componenti del gruppo. Lo studio necessario per svolgere i compiti e utilizzare gli strumenti viene principalmente eseguito in modo individuale e autonomo da un componente del gruppo definito periodicamente dal responsabile di progetto$_G$. Al termine dello studio, egli deve comunicare ciò che ha imparato agli altri membri per garantire che tutti apprendano le nozioni. Inoltre, qualora un componente ritenga di non avere ancora sufficienti capacità o conoscenze per svolgere un compito assegnato, si deve rivolgere al responsabile di progetto$_G$ che ha l'incarico di organizzare un seminario mirato per il suo apprendimento.

4.2.4 Attività

4.2.4.1 Condivisione del materiale

Il materiale del progetto$_G$ è inserito all'interno di un sistema di versionamento$_G$ che permette ai membri del gruppo di integrare le proprie conoscenze e apprendere dal lavoro altrui. Per ogni documento, il materiale utilizzato per la comprensione dei concetti e la stesura è indicato nei "Riferimenti Informativi". Inoltre sono presenti i "Riferimenti Normativi" dentro i quali è presente il documento *Norme di Progetto v. 4.1.1* che tutti devono rispettare.

A Standard di qualità

A.1 ISO/IEC 25010

Lo standard internazionale ISO/IEC 25010 è utilizzato per misurare la qualità del software. Per farlo utilizza una serie di parametri e caratteristiche che possono essere utilizzate per valutare la qualità del software, grazie alle metriche$_G$ fornite dallo standard ISO/IEC 25023.

A.1.1 Metriche di qualità interna

La qualità interna del software è relativa alle attività di progettazione$_G$ e codifica, essa rappresenta la qualità della struttura e del codice del prodotto$_G$. La misurazione della qualità interna viene effettuata principalmente attraverso l'analisi statica$_G$ del codice. Questi controlli sono utili perché semplificano il lavoro dei programmatori e permettono di apportare miglioramenti strutturali al software senza la necessità di eseguirlo. Con un buon livello di qualità interna si ottiene una solida struttura del prodotto$_G$, come base per la qualità esterna e in uso.

A.1.2 Metriche di qualità esterna

La qualità esterna del software è relativa alle proprietà dinamiche e comportamentali del software. Essa viene infatti misurata attraverso l'analisi dinamica, ovvero tramite l'esecuzione del codice e lo svolgimento di test.

A.1.3 Metriche di qualità in uso

La qualità in uso è probabilmente la più difficile da misurare poiché è valutata in un ambito di utilizzo reale, con la partecipazione degli utenti. Essa dipende dalla qualità interna e da quella esterna; infatti per avere una buona qualità in uso è necessario un buon livello di qualità interna ed esterna.

A.1.4 Modello di qualità del prodotto

A.1.4.1 Qualità dei documenti Le metriche utilizzate per misurare la qualità dei documenti sono:

- **M15 Indice di Gulpease**: indica il livello di leggibilità di un testo: più alto questo valore e più è alta la leggibilità del testo. In particolare con valore inferiore a 80 risultata difficile da leggere per chi ha la licenza elementare; inferiore a 60 risultata difficile da leggere per chi ha la licenza media; inferiore a 40 risultata difficile da leggere per chi ha un diploma superiore;

  formula: $89 + \frac{300 \cdot numero\ di\ frasi - 10 \cdot numero\ di\ lettere}{numero\ di\ parole}$;

A.1.4.2 Qualità del software Per descrivere il modello di qualità del prodotto$_G$ software il nostro gruppo ha deciso di utilizzare lo standard ISO/IEC 25010, che determina le caratteristiche della qualità di un prodotto$_G$ software che verranno valutate. Il modello di qualità del prodotto$_G$, definito nello standard, è suddiviso nelle seguenti caratteristiche:

A.1.4.2.1 Idoneità funzionale

Per adeguatezza funzionale si intende la capacità di un prodotto$_G$ software di fornire funzioni che soddisfano i requisiti impliciti ed espliciti, quando usati in un determinato contesto sotto specifiche condizioni.
Le metriche$_G$ sono:

- **M16 Percentuale di requisiti obbligatori soddisfatti**: indica la percentuale di requisiti obbligatori che sono stati soddisfatti. Questo valore deve essere massimizzato per garantire che i vincoli col proponente vengano rispettati;

  formula: $\frac{requisiti\ obbligatori\ soddisfatti}{requisiti\ obbligatori\ totali}$;

- **M17 Percentuale di requisiti desiderabili soddisfatti**: indica la percentuale di requisiti desiderabili che sono stati soddisfatti. Un valore elevato denota una maggior copertura delle richieste del proponente;

  formula: $\frac{requisiti\ desiderabili\ soddisfatti}{requisiti\ desiderabili\ totali}$.

- **M18 Percentuale di requisiti opzionali soddisfatti**: indica la percentuale di requisiti opzionali che sono stati soddisfatti. Un valore elevato denota una maggior copertura delle richieste non indispensabili del proponente;

  formula: $\frac{requisiti\ opzionali\ soddisfatti}{requisiti\ opzionali\ totali}$.

A.1.4.2.2 Affidabilità

Un prodotto$_G$ software è considerato affidabile quando mantiene un livello di prestazioni costante per un determinato intervallo di tempo sotto precise condizioni che possono variare nel tempo.

A.1.4.2.3 Efficienza prestazionale

L'efficienza prestazionale è una caratteristica che indica le prestazioni rispetto al numero di risorse utilizzate sotto condizioni precise. Dato che il nostro progetto si appoggia sul sistema Grafana$_G$ e su algoritmi forniti dal proponente, l'efficienza prestazionale viene delegata a questi due strumenti.

A.1.4.2.4 Sicurezza

Per sicurezza in un prodotto$_G$ software si intende il grado di protezione che il software ha sui dati che manipola per evitare che persone o servizi non autorizzati li ricevano.

- **Confidenzialità**: capacità del software di assicurare la confidenzialità dei dati alle persone che hanno l'autorizzazione di accedere a tali informazioni;
- **Integrità**: capacità del software di evitare modifiche o accessi non autorizzati;
- **Non ripudio**: capacità del software di dimostrare che eventi o azioni sono avvenuti, in modo da non ripudiarli in un secondo momento;
- **Responsabilità**: capacità del software di risalire a una sola entità a partire da una azione di una entità;
- **Autenticità**: capacità del software di provare l'identità delle risorse utilizzate.

A.1.4.2.5 Manutenibilità

Per manutenibilità si intende la capacità di un prodotto$_G$ software di essere modificato e aggiornato senza l'introduzione di problemi e anomalie in seguito a cambiamenti dell'ambiente e/o dei requisiti.

A.1.4.2.6 Portabilità

Per portabilità si intende la caratteristica del software di essere utilizzato in modo efficacie quando esso viene trasferito su diversi hardware o software.

- **Adattabilità**: la capacità del software di essere adattato per hardware o software in continuo cambiamento;
- **Installabilità**: la capacità del software di essere installato o disinstallato efficacemente in un ambiente specifico;
- **Sostituibilità**: la capacità del software di rimpiazzare un altro prodotto$_G$ con lo stesso scopo nello stesso ambiente.

A.1.4.2.7 Usabilità

L'usabilità è una caratteristica di un software definita come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti.