Norme di Progetto

Gruppo TeamAFK - Progetto "Predire in Grafana"

gruppoafk15@gmail.com

Informazioni sul documento

Versione	3.0.0
Approvatore	Victor Dutca
Redattori	Davide Zilio
Verificatori	Simone Federico Bergamin
Uso	Interno
Distribuzione	Prof. Vardanega Tullio Prof. Cardin Riccardo TeamAFK

Descrizione

Questo documento racchiude le regole, gli strumenti e le convenzioni adottate dal *TeamAFK* durante la realizzazione del progetto *Predire in Grafana*.

Registro delle modifiche

Versione	Data	Descrizione	Nominativo	Ruolo
3.0.0	2020-05-26	Approvazione del documento per la RQ	Victor Dutca	*Responsabile*
2.2.1	2020-05-25	Apportate aggiunte a §2.2. Apportate modifiche a §2.2.4.2 e §4.1.4. Verificato il documento.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*
2.2.0	2020-05-24	Refactoring §4. Apportate modifiche normative al *Registro delle Modifiche*. Verificato il documento.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*
2.1.0	2020-05-23	Refactoring §3, aggiunta e modifiche varie. Verificato il documento.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*
2.0.0	2020-05-09	Approvazione del documento per la RP	Victor Dutca	*Responsabile*
1.1.0	2020-04-30	Aggiunte sezioni §2.1.4, §2.4, §3.3, §3.11 §4.4 e §A. Correzioni varie. Aggiunti strumenti §2.3. Aggiunti riferimenti §3.10. Refactor §1.4, §3.12. Verificato il documento.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*
1.0.2	2020-04-29	Correzione *RdM*, conformati titoli e accenti. Corretti errori ortografici. Verificato il documento.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*
1.0.1	2020-04-21	Aggiunti strumenti e verifica §2.3, §2.4.	Davide Zilio Simone Federico Bergamin	*Amministratore* *Verificatore*

1.0.0	2020-03-27	Approvazione del documento	Simone Federico Bergamin	*Responsabile*
0.3.9	2020-03-26	Aggiornamento e verifica §2.2 - §2.4.	Olivier Utshudi Fouad Farid	*Amministratore* *Verificatore*
0.3.8	2020-03-26	Aggiornamento e verifica §3.10.	Davide Zilio Fouad Farid	*Amministratore* *Verificatore*
0.3.7	2020-03-26	Aggiornamento e verifica §4.1 - §4.3.	Simone Meneghin Alessandro Canesso	*Amministratore* *Verificatore*
0.3.6	2020-03-25	Apportate correzioni e verifica §3.	Davide Zilio Alessandro Canesso	*Amministratore* *Verificatore*
0.3.5	2020-03-25	Apportate correzioni e verifica §4.	Simone Meneghin Fouad Farid	*Amministratore* *Verificatore*
0.3.4	2020-03-25	Apportate correzioni e verifica §2.	Olivier Utshudi Fouad Farid	*Amministratore* *Verificatore*
0.3.3	2020-03-24	Apportate aggiunte e verifica §2.2 - §2.3.2.2.	Olivier Utshudi Alessandro Canesso	*Amministratore* *Verificatore*
0.3.2	2020-03-24	Apportate aggiunte e verifica §4.2.2 - §4.3.	Simone Meneghin Alessandro Canesso	*Amministratore* *Verificatore*
0.3.1	2020-03-24	Apportate aggiunte e verifica §3.6.3 - §3.8.4.	Davide Zilio Fouad Farid	*Amministratore* *Verificatore*
0.3.0	2020-03-23	Stesura e verifica §1 e §2.	Olivier Utshudi Fouad Farid	*Amministratore* *Verificatore*
0.2.0	2020-03-23	Stesura e verifica §4.	Simone Meneghin Alessandro Canesso	*Amministratore* *Verificatore*
0.1.0	2020-03-23	Stesura e verifica §3	Davide Zilio Alessandro Canesso	*Amministratore* *Verificatore*

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il seguente documento ha il compito di stabilire le regole che il team di sviluppo intende seguire in ogni attività di progetto, così da omologare il materiale prodotto. I fornitori intendono adottare un approccio incrementale$_G$, affinché lo sviluppo del prodotto si basi su decisioni prese di comune accordo. Perciò ogni componente del team deve far riferimento a questo documento per garantire la coesione e uniformità delle scelte prese.

1.2 Scopo del prodotto

Il capitolato C4$_G$ illustra il prodotto da fornire. Tale prodotto consiste in tool di addestramento ed un plugin$_G$ di Grafana$_G$ che prenderà il nome *Predire in Grafana*. Entrambi gli applicativi verranno scritti in linguaggio JavaScript$_G$. Il primo avrà il compito di produrre un file di estensione JSON$_G$ basato su dati di addestramento$_G$. Il secondo invece si occuperà di leggere il file creato, effettuare previsioni basandosi su di esso e rendere disponibili al sistema i risultati ottenuti, in modo da poterli visualizzare in grafici e dashboard.

1.3 Glossario

Per evitare ambiguità nei documenti formali, viene fornito il documento *Glossario_v2.0.0*, contenente tutti i termini considerati di difficile comprensione. Perciò nella documentazione fornita ogni vocabolo contenuto in *Glossario_v2.0.0* è contrassegnato dalla lettera *G* a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

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

1.4.2 Riferimenti informativi

- **Standard ISO/IEC 12207:1995**:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- **Change Management Process**:
  https://www.blog-management.it/change-management-project-management
  https://www.digital4.biz/hr/hr-transformation/;
  - **The three P's of Software Engineering**:
    http://dwaynephillips.net/CutterPapers/ppp/ppp.htm;
- **Software Engineering - Ian Sommerville - 10th Edition**
  Capitoli di riferimento:
  - §2 - Processi Software;

  - §3 - Sviluppo agile del software;
  - §4 - Ingegneria dei requisiti;
  - §8 - Test del software;
  - §9 - Evoluzione del software;
  - §19 - Gestione della progettazione;
  - §21 - Gestione della qualità;
  - §22 - Gestione della configurazione.
- **Slide L05 del corso Ingegneria del Software - Ciclo di vita del software**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L05.pdf;
- **Slide L06 del corso Ingegneria del Software - Gestione di Progetto**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L06.pdf;
- **Slide L12 del corso Ingegneria del Software - Qualità di Prodotto**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L12.pdf;
- **Slide L13 del corso Ingegneria del Software - Qualità di Processo**: https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L13.pdf.

2 Processi primari

2.1 Fornitura

2.1.1 Scopo

Lo scopo del processo di fornitura consiste nelle attività e compiti dell'acquirente, come risposta ai bisogni del cliente in ambito di sistema, prodotto e/o servizio software. Nello specifico, le caratteristiche richieste dal proponente sono analizzate e stilate nello *Studio di Fattibilità*, il quale è alla base del processo di fornitura. Dopodiché è possibile stabilire le risorse e le procedure necessarie alla redazione di un *Piano di Progetto* da seguire fino alla consegna del materiale prodotto. Nel dettaglio le attività svolte da questo processo sono:

- avvio;
- studio di fattibilità;
- contrattazione;
- progettazione;
- esecuzione e controllo;
- revisione e valutazione;
- consegna e completamento.

2.1.2 Aspettative

Il gruppo deve avere un frequente contatto con il cliente al fine di rispettare i vincoli obbligatori da lui fissati, onde evitare di scostarsi troppo dal prodotto finale richiesto. Per questo è necessario comunicare con il proponente, per poter aver chiari i bisogni che tale prodotto intende soddisfare con le sue funzionalità. Quindi, ogniqualvolta il gruppo venga a contatto con il proponente, deve stabilire:

- aspetti chiave che soddisfano i bisogni richiesti;
- requisiti e vincoli dei processi;
- verifica continua;
- chiarimento di eventuali dubbi;
- accordo sulla qualifica del prodotto.

2.1.3 Descrizione

Questa sezione ha il compito di mostrare le norme che il *TeamAFK* adotterà in tutte le attività di progettazione, sviluppo e consegna del prodotto *Predire in Grafana*, con lo scopo di diventare i fornitori del capitolato proposto dalla proponente *Zucchetti SPA* e dai committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin.

2.1.4 Attività

2.1.4.1 Avvio

Studio di Fattibilità

Lo *Studio di Fattibilità*, redatto per ogni capitolato$_G$ dagli analisti, illustra:

- **Descrizione generale**: sintesi delle informazioni generali del capitolato e delle circostanze da cui sorgono tali richieste;
- **Obiettivi**: vengono mostrate le caratteristiche principali del prodotto ed i relativi obiettivi da raggiungere;
- **Tecnologie utilizzate**: corrisponde all'elenco degli strumenti messi a disposizione dall'azienda proponente al team per sviluppare il progetto;
- **Valutazione finale**: si tratta di un commento elaborato dal gruppo una volta riunitosi e analizzato le richieste dell'azienda, motivando la scelta di focalizzarsi o meno su una determinata offerta.

2.1.4.2 Contrattazione

L'attività di contrattazione prevede la discussione dei termini di sviluppo del progetto sia per quanto concerne l'aspetto economico, sia per quanto riguarda l'aspetto dei requisiti che il prodotto finale deve rispettare. Come base di discussione il *TeamAFK* propone il preventivo all'interno del *Piano di Progetto* e la lista dei requisiti esposti all'interno dell'*Analisi dei Requisiti*. Il gruppo si impegna ad aggiornare tali valutazioni proposte sulla base dei feedback e della discussione con il committente.

2.1.4.3 Progettazione

L'attività di progettazione prevede la redazione di un *Piano di Progetto* che definisca le linee guida per la gestione e la realizzazione del progetto e che garantisca il raggiungimento degli obiettivi entro i termini prefissati.

Coinvolgimento di committente e proponente

Il *TeamAFK* prevede di coinvolgere il proponente Gregorio Piccoli - *Zucchetti SpA* nell'analisi dei requisiti e nello sviluppo del progetto tramite incontri su appuntamento. Il coinvolgimento del committente è invece garantito dalla presenza di revisioni periodiche, come previsto dalle regole di progetto.

Piano di Progetto

Il *Piano di Progetto* è un documento redatto dal project manager$_G$ e dagli amministratori; si aggiunge all'insieme dei documenti che il team dovrà seguire per tutta la durata del progetto. In particolare questo documento conterrà:

- **Analisi dei rischi**: il *Responsabile* analizza tutti gli eventuali rischi di tipo tecnico, economico e temporale che possono presentarsi durante il progetto, fornendo anche soluzioni che possano risolverli o almeno limitare i loro effetti negativi;
- **Modello di sviluppo**: viene definita la struttura su cui basarsi per la pianificazione, esecuzione e consegna del prodotto software;
- **Pianificazione**: viene pianificato l'insieme delle attività da seguire durante le fasi di progetto, collocandole nel tempo e stabilendo le loro scadenze;
- **Preventivo e consuntivo**: con il preventivo viene mostrata una stima di quello che sarà il carico di lavoro che il team sosterrà durante il progetto, in termini di tempo e i relativi costi. Con il consuntivo di periodo, invece verranno riportati le variazioni dei costi rispetto a quanto preventivato.

2.1.4.4 Esecuzione e controllo

Il fornitore deve attuare ed eseguire i piani sviluppati nella fase di pianificazione e sviluppare il prodotto software monitorando il progresso e la qualità.

Piano di Qualifica

I verificatori si occuperanno di redigere il *Piano di Qualifica*, ovvero l'insieme di attività con il compito di fissare obiettivi di qualità del prodotto nel suo complesso. Verranno quindi considerati anche i processi e le risorse necessarie a raggiungere tali obiettivi. Nel dettaglio il *Piano di Qualifica* si focalizza su:

- **Qualità di prodotto**: vengono fissate le politiche per il raggiungimento della qualità, gli obiettivi da raggiungere e gli strumenti necessari al controllo;
- **Qualità di processo**: stabilire sulla base di opportune misurazioni il grado di efficacia ed efficienza di un processo a partire dalla sua definizione;
- **Valutazione di miglioramento**: i problemi e le relative soluzioni vengono evidenziate in questa sezione del *Piano di Qualifica*;
- **Resoconto dell'attività di verifica**: vengono riportate i risultati delle metriche come resoconto dell'attività di qualifica.

Standard di qualità

La sezione §A descrive gli standard di qualità di riferimento per lo sviluppo della qualità del progetto.

2.1.4.5 Revisione e valutazione

Il fornitore deve eseguire la verifica e validazione al fine di dimostrare che il prodotto software soddisfi i requisiti descritti nell'*Analisi dei Requisiti*.

2.1.4.6 Consegna e completamento

Il fornitore consegna il prodotto software come specificato dal contratto.

2.1.5 Metriche di qualità

Per questo processo non sono state definite metriche di qualità specifiche.

2.1.6 Strumenti di supporto

Di seguito sono illustrati gli strumenti utilizzati per l'avvio dell'attività di fornitura.

- **GanttProject**: software per la produzione di diagrammi di Gantt$_G$ (analizzato nel dettaglio in § 3.8.3);
- **Google Calendar**: sistema di calendari gestito da Google.

2.2 Sviluppo

2.2.1 Scopo

È il processo che si occupa di stabilire le attività da svolgere per costruire e consegnare il prodotto finale.

2.2.2 Aspettative

Le aspettative sono le seguenti:

- stabilire obiettivi di sviluppo;
- stabilire vincoli tecnici;
- stabilire vincoli di design;
- realizzare il prodotto software che superi i test, e soddisfi i vincoli di progetto.

2.2.3 Descrizione

Il processo di sviluppo si divide in:

- analisi dei requisiti;
- progettazione;
- codifica.

2.2.4 Attività

2.2.4.1 Analisi dei Requisiti

L'elenco dei requisiti necessari allo svolgimento del processo di sviluppo viene raccolto e

redatto dagli analisti in un apposito documento di *Analisi dei Requisiti*. Quest'ultimo ha lo scopo di:

- descrivere l'obiettivo del prodotto;
- fornire ai progettisti riferimenti precisi ed affidabili;
- stabilire le prospettive e le funzionalità del prodotto in base ai vincoli fissati dal proponente;
- fornire ai verificatori riferimenti affidabili per la loro attività di controllo;
- valutare rischi, costi e benefici in relazione al carico di lavoro.

2.2.4.1.1 Aspettative

L'obiettivo di questa attività è redigere un documento formale contenente tutti i requisiti richiesti dal progetto.

2.2.4.1.2 Descrizione

I requisiti saranno raccolti nel seguente modo:

- lettura del capitolato d'appalto;
- confronto con il proponente;
- discussione tra i componenti del gruppo;
- studio dei casi d'uso.

2.2.4.1.3 Casi d'uso

I casi d'uso sono scenari che descrivono una possibile sequenza di iterazioni dell'utente, visto come attore$_G$ attivo e/o passivo, e il sistema. La struttura di un caso d'uso è la seguente:

- codice identificativo;
- titolo;
- diagramma UML$_G$ (se presente);
- attore primario;
- precondizioni;
- postcondizioni;
- scenario principale;
- inclusioni (se presenti);
- estensioni (se presenti).

Codice identificativo dei casi d'uso

Il codice di ogni caso d'uso seguirà questo formalismo:

UC[codice_padre].[codice_figlio]

Requisiti

I requisiti seguiranno la seguente struttura:

- **codice identificativo**: è un codice univoco e conforme alla codifica:

  Re[Importanza][Tipologia][Codice]

  Le voci riportate nella precedente codifica significano:

  - **Importanza**: la quale può assumere come valori:
    - 1: requisito obbligatorio, irrinunciabile;
    - 2: requisito desiderabile, perciò non obbligatorio ma riconoscibile;
    - 3: requisito opzionale, ovvero trattabile in un secondo momento o relativamente utile.
  - **Tipologia**: la quale può assumere come valori:
    - F: requisito funzionale;
    - P: requisito prestazionale;
    - Q: requisito di qualità;
    - V: requisito di vincolo.
  - **Codice identificativo**: il quale è un identificatore univoco del requisito, e viene espresso in forma gerarchica padre/figlio.
- **Classificazione**: specifica il peso del requisito facilitando la sua lettura anche se causa ridondanza;
- **Descrizione**: sintesi completa di un requisito;
- **Fonti**: il requisito può avere le seguenti provenienze:
  - capitolato;
  - interno: requisito che gli analisti ritengono di aggiungere in base alle esigenze del team;
  - caso d'uso: il requisito proviene da uno o più casi d'uso, dei quali è necessario riportare il codice univoco di caso d'uso;

- verbale: dopo un chiarimento da parte del proponente è possibile che sorga un requisito non preventivato. E le informazioni su di esso sono riportate e tracciati nei rispettivi verbali.

UML

I diagrammi UML servono per descrivere un caso d'uso. Gli analisti dovranno utilizzare la versione v.2.0.

2.2.4.2 Progettazione

2.2.4.2.1 Scopo

Il processo di progettazione ha il compito di stabilire le migliori operazioni da effettuare per fornire una soluzione soddisfacente per tutti gli stakeholders$_G$. Per fare ciò è necessario identificare la struttura complessiva del sistema, per poi suddividerla nei moduli e unità architetturali che la compongono. In questo modo è più semplice per i programmatori che implementeranno il sistema, comprendere e rispettare i requisiti funzionali. A questo scopo si deve:

- tradurre i requisiti in unità di codice (i moduli);
- assegnare ai programmatori singoli compiti;
- fornire un prototipo di sistema da migliorare, ma funzionante;
- garantire il tracciamento dei requisiti per componente.

Strategie adottate:

- suddivisione della macro-architettura;
- implementazione dei moduli.

2.2.4.2.2 Descrizione

Il *TeamAFK* adotta un approccio **agile** allo sviluppo del prodotto software, lasciando immutate le esigenze di documentazione correlata con la gestione dell'avanzamento di progetto e le verifiche di qualità.
Lo svilppo dell'architettura software si suddivide in due momenti specifici:

- **Technology baseline**: mostra ad alto livello le specifiche di progettazione del prodotto e le sue componenti, elencando i diagrammi UML utilizzati per descrivere l'architettura del prodotto. Viene realizzata una Proof of Concept$_G$ del prodotto;
- **Product baseline**: parte incrementale del prodotto finale da cui continuare a lavorare, integrando le specifiche riportate nella Tecnology Baseline e definendo anche i test necessari alla verifica. Viene presentata tramite diagrammi delle classi e di sequenza. L'analisi è comprensiva di design pattern contestualizzati all'architettura.

Technology Baseline

Il progettista incaricato, si occuperà di includere:

- **Diagrammi UML**: utilizzati per rappresentare i casi d'uso descritti nell'*Analisi dei Requisiti*;
- **Tecnologie utilizzate**: vengono mostrate le tecnologie impiegate, mostrando le loro funzionalità, pregi e difetti;

- **Proof of Concept (POC)**: implementazione di un PoC che utilizzi e dimostri la fattibilità delle tecnologie che utilizziamo per implementare il prodotto software;
- **Tracciamento delle componenti**: viene mostrata la relazione tra il componente e il requisito che intende rispettare;
- **Test di integrazione**: sono operazioni che si occupano di verificare l'unione tra le parti, in base alle rispettive interfacce.

Proof of Concept

Il *TeamAFK* ha deciso di implementare un PoC in modo incrementale. Perciò, in accordo con il proponente, vengono selezionati degli incrementi da sviluppare con l'intenzione di esplorare tutte le tecnologie che devono essere utilizzate nel prodotto finale. Queste tecnologie sono:

- Node.js;
- React;
- SVM;
- JavaScript;
- TypeScript;
- JSON;
- HTML5;
- CSS3;
- Grafana;
- Chart.js.

Product Baseline

Gli aspetti su cui la Product baseline si sofferma sono:

- **Definizione delle classi**: ogni classe deve essere descritta, specificando scopo e funzionalità;
- **Tracciamento delle classi**: ovvero identificare il requisito a cui si lega una classe;
- **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. I diagrammi UML utilizzati per presentare l'architettura sono i seguenti:
  - **Diagramma delle classi**: i diagrammi delle classi descrivono in modo dettagliato gli oggetti che utilizziamo e le dipendenze che sussistono tra di loro;
  - **Diagramma dei package**: con i diagrammi dei package raggruppiamo ad alto livello i macro insiemi delle nostre componenti, suddividendole in package. Questi

ultimi infatti individuano uno spazio dei nomi che tra loro non presentano alcuna ambiguità. Inoltre, grazie ai diagrammi dei package, definiamo le dipendenze ad alto livello tra le componenti; ciò è molto utile per comprendere la complessità strutturale ed il flusso delle dipendenze stesse. Permettono infatti di individuare circolarità ed elevata complessità delle dipendenze in modo immediato;

  - **Diagramma di attività**: i diagrammi di attività permettono di descrivere un processo attraverso dei grafi, in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite;
  - **Diagramma di sequenza**: i diagrammi di sequenza descrivono la collaborazione di un gruppo di oggetti che devono implementare collettivamente un comportamento, nello scorrere del tempo.

- **Design pattern**$_G$: sono esposti i design pattern adottati per rappresentare l'architettura del prodotto. Tutti i design pattern devono essere accompagnati da un diagramma e una descrizione delle sue caratteristiche;
- **Test di integrazione**: test necessari per verificare che l'unione delle parti funzioni correttamente;
- **Test di unità**: ovvero verificare le funzionalità della sola classe, senza metterla in relazione con altre componenti del sistema.

2.2.4.2.3 Qualità dell'architettura

In seguito alla specifica dei requisiti che consolida le funzionalità richieste, viene realizzata l'architettura del sistema. Tale architettura deve perseguire le seguenti caratteristiche:

- appropriatezza: l'architettura soddisfa tutti i requisiti;
- modularità: l'architettura è suddivisa in parti ben distinte;
- robustezza: l'architettura può sopportare diversi ingressi dall'utente e dall'ambiente;
- affidabilità: l'architettura garantisce una buona usabilità del prodotto quando viene implementata;
- manutenibilità: l'architettura può subire modifiche a costi contenuti.

In particolare, riguardo alla modularità, il progettista deve scomporre il sistema in moduli, fornendo una descrizione precisa della struttura modulare e delle relazioni che esistono tra essi. Questo porta ai seguenti vantaggi:

- maggiore leggibilità e riusabilità del codice;
- semplificazione dell'individuazione e della correzione degli errori;
- realizzazione di prototipi (in relazione al modello di sviluppo adottato).

Per perseguire le qualità sopra elencate, è richiesta l'implementazione di design pattern ove necessario.

Al fine di implementare le qualità dell'architettura appena descritte, ai progettisti viene richiesto di seguire le seguenti regole:

- evitare package vuoti, classi e parametri non utilizzati e parametri senza tipo;
- evitare la presenza di dipendenze circolari;
- evitare modifiche ad eventuali librerie esterne utilizzate;
- utilizzare nomi significativi per le classi e i metodi in esse contenuti, in modo da facilitare la comprensione immediata di ogni componente.

2.2.4.3 Codifica

Scopo

Lo scopo del processo di codifica è implementare il prodotto software richiesto. Il programmatore è colui che ha il compito di attuare i design pattern e non può agire diversamente.

Aspettative

L'obiettivo di questo processo è la costruzione del prodotto richiesto secondo le specifiche richieste dal proponente. Perciò il programmatore deve attenersi alle norme qui stabilite, al fine di produrre un codice solido e uniforme, facilitando l'attivita di manutenzione, verifica, validazione e miglioramento della qualità del prodotto.

Descrizione

Il codice deve attenersi alle norme e rispettare i requisiti di qualità, espressi nel documento *Piano di Qualifica*, per garantirne la qualità desiderata.

Stile di codifica

Lo stile di codifica stabilisce le norme che il programmatore deve rispettare, focalizzandosi nei seguenti ambiti:

- **Indentazione**: blocchi di codice innestato deve avere per ogni livello di indentazione quattro spazi, ad eccezione dei commenti. È consigliato impostare adeguatamente la configurazione dell'IDE$_G$ usato:

```
function() {
    let x = 0;
}
```

- **Parentesizzazione**: si richiede di aprire le parentesi in linea e non al di sotto dei costrutti a cui si riferiscono:

```
class Example {
    // TO-DO
}
```

- **Scrittura dei metodi**: è desiderabile la poca prolissità del codice di un metodo;
- **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
let myVar1 = 1;
let myVar2 = 2;
let myVar3 = myVar1 + myVar2;
```

- **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
}
```

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

```
// NO
let x = 1, y = 2;
// OK
let x = 1;
let y = 2;
```

- **Univocità dei nomi**: per evitare ambiguità e incomprensioni, tutti i nomi di classi, metodi, funzioni ed interfacce, devono essere univoci ed esplicativi;
- **Classi**: i nomi delle classi devono iniziare con la lettera maiuscola:

```
class Example {
   // TO-DO
}
```

- **Costanti**: i nomi delle costanti devono essere scritte con lettere maiuscola:

```
const CONSTANT_VALUE = "This is a constant value";
```

- **Metodi**: i nomi dei metodi devono iniziare con la lettera minuscola, seguita da lettere minuscole. Nel caso di nomi composti da più parole, tutte quelle che seguono la prima iniziano con la prima lettera maiuscola a cui seguono lettere minuscole. Viene così rispettato il modello *CamelCase*$_G$:

```
class MyClass {
   exampleMethod() {
      //...
   }
}
```

- **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:

```
let varExample = 0;
```

- **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 `//**...*/`. Un esempio è il seguente:

```
//** This is a comment class example.
* OK!
*/
class myClass {
   // This is a test method.
   myMethod() {
      let myVar = 0;
      if (condition) {
         //TO-DO
      } else {
         //TO-DO
      }
   }
}
```

- **Lingua**: codice e commenti devono essere espressi in lingua inglese.

Ricorsione

La ricorsione deve essere il più possibile evitata, perché è causa di un aumento di complessità della soluzione ad un problema. Per questo è preferibile adottare soluzioni iterative.

2.2.5 Metriche di qualità

2.2.5.1 Documenti

Le metriche utilizzate per determinare la qualità e leggibilità dei documenti sono descritte di seguito:

- **Indice di Gulpease**: indice di leggibilità di un testo tarato sulla lingua italiana e basato su due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere;
- **Indice Fog**: misura la lunghezza media delle parole e delle frasi nei documenti. Quanto più è alto il valore dell'indice Fog, tanto più è difficile capire il documento.

Una loro descrizione più accurata la si trova nella sezione §3.1.5.

Analisi dei Requisiti

Durante il periodo di analisi vengono identificati i requisiti del progetto e definiti i relativi casi d'uso. Gli obiettivi sono:

- formulare la definizione di casi d'uso e requisiti;
- ottenere la loro approvazione;
- tracciare il loro cambiamento nel tempo.

La strategia prevede:

- considerare lo scopo del progetto e le richieste degli stakeholder;
- esprimere ciò in forma di requisiti, classificati in obbligatori, desiderabili e opzionali;
- valutare il corpo dei requisiti e negoziare cambiamenti se necessario;
- ottenere la loro approvazione da parte del proponente.

Le metrice utilizzate per la verifica della qualità dei requisiti sono:

- **MG02 - PROS**;
- **MG03 - PRDS**.

Quest'ultime sono descritte nella sezione §3.3.5.

2.2.5.2 Progettazione

Questo paragrafo elenca le metriche a cui l'architettura software viene sottoposta per valutarne la qualità.

Tabella 2.2.1: Metriche per la qualità dell'architettura software

Nome	Codice	Descrizione
Fan-In	MS05	Misura il numero di funzioni o di metodi che chiamano una funzione o un metodo X. Un alto valore implica che X è strettamente collegata al resto del progetto e che eventuali modifiche di X produrranno ampi effetti a catena.
Fan-Out	MS06	Misura il numero di funzioni che sono chiamate dalla funzione X. Un alto valore indica che la complessità generale di X può essere alta a causa della complessità della logica di controllo necessaria per coordinare i componenti chiamati.

2.2.5.3 Codifica

Questo paragrafo elenca le metriche a cui il prodotto software viene sottoposto per misurarne la qualità. I valori riportati sono una dichiarazione di intenti; nulla vieta che in corso d'opera possano avvenire delle revisioni.

Tabella 2.2.2: Metriche del software

Nome	Codice	Descrizione
Linee di Codice (LOC)	MS01	Rappresenta le dimensioni del codice di un metodo. È espresso in termini di numero di linee di codice, ed è utilizzato per dimensionare la produttività delle persone e da questa lo sforzo richiesto per sviluppare tale funzione.
Numero di Metodi (NM)	MS02	Numero medio di metodi contenuti nelle classi di un oggetto. Un numero troppo alto di metodi può indicare la necessità di scomporre la classe. Un numero troppo basso deve far riflettere sull'effettiva utilità della classe in esame. $$NM = \frac{\sum \#metodi}{\#classi}$$
Numero di Parametri (NP)	MS03	Numero di parametri passati ad un metodo. Un eccessivo numero di parametri passati ad un metodo può indicare un'eccessiva complessità dello stesso.
Commenti per Linee di Codice (CLC)	MS04	Rapporto fra numero di righe di commento e numero totale di righe (vuote escluse). Un codice ben commentato può essere compreso più facilmente e velocemente, facilitando le operazioni di manutenzione. $$CLC = \frac{\#righe\_commento}{\#tot\_righe}$$

2.2.6 Strumenti di supporto

2.2.6.1 Browser

Per provare e verificare il funzionamento del nostro tool e plug-in usiamo i seguenti broswer:

- **Google Chrome**: versione 58 o successiva;
- **Internet Explorer**: versione 11;
- **Microsoft Edge**: versione 14 o successiva;
- **Mozilla Firefox**: versione 54 o successiva;
- **Safari**: versione 10 o successiva..

2.2.6.2 Coveralls

Servizio web utilizzato per tenere traccia della copertura del codice.
https://coveralls.io/

2.2.6.3 Draw.io

Programma utilizzato per la creazione di diagrammi UML.
https://app.diagrams.net/

2.2.6.4 ESLint

È un plugin$_G$ utilizzato per effettuare analisi statica del codice e rilevare problematiche nei pattern codificati in linguaggio JavaScript$_G$.
https://eslint.org/

2.2.6.5 Grafana

Software di analisi open source utilizzato per visualizzare e monitorare i dati attraverso grafici e avvisi. Grafana viene fornito con un plug-in per InfluxDB. Quest'ultimo include un editor di query personalizzato e supporta annotazioni e modelli di query.
https://grafana.com/

2.2.6.6 InfluxDB

Database$_G$ di serie temporali open source utilizzato per la gestione e l'aggiornamento dei dati.
https://www.influxdata.com/

2.2.6.7 IntelliJ IDEA

IDE$_G$ utilizzato per la codifica in JavaScript, garantendo la piena compatibilità con i sistemi

operativi Linux, Windows e MacOS.
https://www.jetbrains.com/idea/

2.2.6.8 Node.js

Utilizzato per creare applicazioni di rete scalabili non solo lato client ma anche lato server.
https://nodejs.org/it/about/

2.2.6.9 NPM

Utilizzato per poter installare i pacchetti necessari allo sviluppo dell'applicativo (React$_G$).
https://www.npmjs.com/

2.2.6.10 React

Per la realizzazione dell'interfaccia utente a livello di applicazione web.
https://reactjs.org/

2.2.6.11 TravisCI

Impiegato per garantire la continuous integration.
https://travis-ci.org/

2.3 Procedure

2.3.1 Draw.io

Creazione dei diagrammi UML

Per la creazione dei diagrammi UML è necessario accedere a Google Drive con le credenziali del gruppo, entrare nella cartella `Analisi_dei_requisiti/img/Diagrammi_UML`.
Un diagramma UML deve essere creato rispettando il seguente ordine:

- nel menù `New`, scegliere `More` e quindi fare click su `draw.io Diagrams`;
- realizzare il diagramma;
- nel menù `File`, scegliere `Export as..` e selezionare `PNG`;
- inserire un nome esplicativo, indicando il codice del caso d'uso di riferimento e un breve titolo;
- alla richiesta di selezionare in quale cartella salvare il diagramma in formato PNG selezionare `No, pic folder..` e assicurarsi di salvare nella cartella apposita denominata `Diagrammi_UML`.

2.3.2 NPM

Installazione di un pacchetto

Per poter installare un pacchetto Node all'interno dell'ambiente di sviluppo IntelliJ, bisogna utilizzare il seguente comando:

```
npm install <nome_pacchetto>
```

Successivamente, provvederà direttamente IntelliJ a scaricare ed importare nel progetto il pacchetto desiderato.

Avvio del server

Per avviare il `server node.js`, bisogna utilizzare il comando `npm start`.
Viceversa, per bloccare la sua esecuzione, utilizzare il comando `npm stop`.

2.3.3 InfluxDB

Innanzittutto è necessario scaricare l'eseguibile per poter avviare il server. Quest'ultimo lo si può scaricare dal seguente link:

https://portal.influxdata.com/downloads/.

Avvio del server

Per avviare il server di InfluxDB, eseguire `influxd.exe`.
Per avviare la CLI, eseguire `influx.exe`.

3 Processi di supporto

3.1 Documentazione

3.1.1 Scopo

Ogni processo$_G$ e attività$_G$ significativi volti allo sviluppo del progetto sono documentati. Lo scopo di questa sezione è definire gli standard che riguardano i documenti prodotti durante l'intero ciclo di vita del software. I documenti sono consultabili nelle relative sezioni della repository$_G$: https://github.com/teamafkSWE/PredireInGrafana-docs.

3.1.2 Aspettative

Ciò che ci aspettiamo dal processo di documentazione è costruire una Body of Knowledge$_G$ che raccoglie la conoscenza in modo sistematico, disciplinato, quantificabile. Quest'ultima deve essere:

- completo;
- non ambiguo;
- modulare;
- coeso;
- trasparente;
- semplice;
- disponibile.

In poche parole, deve permettere la trasmissione delle informazioni nel miglior modo possibile.

3.1.3 Descrizione

Questo processo descrive come il gruppo ha gestito il processo di documentazione utile ai fini del progetto. Perciò si definiscono le fasi del ciclo di vita specificando come i documenti vengono costruiti, strutturati e redatti.

3.1.4 Attività

3.1.4.1 Ciclo di vita di un documento

Ogni documento segue le seguenti fasi di ciclo di vita:

- **Sviluppo**: creazione del documento, definizione della struttura e prima stesura di tutte le parti che lo compongono;
- **Verifica**: un documento entra in fase di verifica successivamente al suo completamento. È dovere del *Responsabile* assegnare tale compito ad almeno un *Verificatore*. Quest'ultimo deve applicare le procedure di verifica e segnalare eventuali modifiche da apportare al documento;

- **Approvazione**: il *Responsabile* approva il documento, che sarà quindi ritenuto completo e pronto per il rilascio;
- **Rivisitazione e ampliamento**: con l'avanzare del progetto si prevede di espandere ciascun documento, aggiungendo nuove sezioni o migliorando quanto scritto in precedenza. Sarà compito del *Responsabile* istanziare una nuova fase di sviluppo per provvedere alla realizzazione di questi aggiornamenti. Al termine di essa vengono eseguite nuovamente le fasi di verifica ed approvazione del documento.

3.1.4.2 Template

È stato creato un template LaTeX per uniformare la struttura grafica e lo stile di formattazione dei documenti. Lo scopo dei template è permettere, a colui che redige il documento, di adottare automaticamente le conformità previste dalle *Norme di Progetto*. Nel caso quest'ultime cambiassero, essi permettono di agevolare la procedura di adeguamento alle nuove norme.

3.1.4.3 Struttura dei documenti

Un file "nome_file.tex" (in cui "nome_file" verrà sostituito dal nome del documento) raccoglie, tramite comandi di input, le sezioni di cui è composto il documento. Tra i file in input ci sono:

- "AFKstyle.sty", contenente i pacchetti necessari alla compilazione e i comandi relativi all'impostazione grafica;
- "copertina.tex", che contiene i comandi LaTeX per l'impostazione della prima pagina del documento;
- "registroModifiche.tex", contenente la tabella delle modifiche.

3.1.4.3.1 Prima pagina

Il frontespizio è la prima pagina del documento ed è così strutturato:

- **Logo del gruppo**: logo del *TeamAFK* visibile come primo elemento centrato in alto;
- **Titolo**: nome del documento, posizionato centralmente sotto il logo;
- **Gruppo e progetto**: nome del gruppo e del progetto *Predire in Grafana*, visibile centralmente sotto il titolo;
- **Recapito**: indirizzo di posta elettronica del gruppo, posizionato sotto il nome del gruppo e del progetto;
- **Informazioni sul documento**: tabella posizionata al di sotto del recapito, contenente le seguenti informazioni:
  - **Versione**: versione del documento;

  - **Approvatore**: nome e cognome dei membri del gruppo incaricati dell'approvazione del documento;
  - **Redattori**: nome e cognome dei membri del gruppo incaricati della redazione del documento;
  - **Verificatori**: nome e cognome dei membri del gruppo incaricati della verifica del documento;
  - **Uso**: tipologia d'uso del documento, che può essere "interno" o "esterno";
  - **Distribuzione**: destinatari del documento.
- **Descrizione**: descrizione sintetica del documento, posizionata centralmente in fondo alla pagina.

3.1.4.3.2 Registro delle Modifiche

Ogni documento dispone di un *Registro delle Modifiche*: una tabella posta a seguito della prima pagina, contenente le modifiche apportate al documento. In essa sono indicati:

- versione del documento dopo la modifica;
- data della modifica;
- breve descrizione della modifica;
- nominativo di chi ha modificato;
- ruolo di chi ha modificato.

3.1.4.3.3 Indice

L'indice ha lo scopo di riepilogare e dare una visione macroscopica della struttura del documento, mostrando le parti gerarchiche di cui è composto. Ogni documento è corredato dall'indice dei contenuti, posizionato dopo il *Registro delle Modifiche*. Se sono presenti immagini o tabelle all'interno del documento, l'indice dei contenuti è seguito prima dalla lista delle immagini, poi dalla lista delle tabelle.

3.1.4.3.4 Contenuto principale

La struttura delle pagine di contenuto è così definita:

- **Logo**: presente in alto a sinistra;
- **Nome del documento**: presente in alto a destra;
- **Riga di separazione**: divide l'intestazione dal contenuto;
- **Contenuto della pagina**: posto tra l'intestazione e il piè di pagina;
- **Riga di separazione**: divide il contenuto dal piè di pagina;

- **Nome e versione del documento**: posto in basso a sinistra;
- **Numero della pagina**: presente in basso a destra, con il formato "Pagina X di Y", in cui la X indica il numero della pagina corrente e Y il numero totale delle pagine.

3.1.4.3.5 Verbali

I verbali vengono prodotti dal/i soggetto/i incaricato/i della loro stesura in occasione di incontri tra i membri del team, con o senza la presenza di referenti esterni. È prevista la stesura di più verbali, uno per ogni incontro. La struttura è così definita:

- **Luogo**: luogo di svolgimento dell'incontro;
- **Data**: data dell'incontro, nel formato `YYYY-MM-DD`;
- **Ora di inizio**: l'orario di inizio dell'incontro;
- **Ora di fine**: l'orario di fine dell'incontro;
- **Partecipanti**: elenco dei membri del gruppo presenti all'incontro e, se presenti, i nominativi delle persone esterne che vi hanno partecipato;
- **Topic**: argomenti affrontati durante l'incontro.

Ogni verbale dovrà essere denominato secondo il seguente formato:

VX_YYYY-MM-DD

dove con "X" bisognerà indicarne la tipologia:

- **I**: verbale "interno", incontro tra i membri del team di progetto;
- **E**: verbale "esterno", incontro con partecipanti esterni al gruppo (committente o proponente), per chiarimenti riguardanti il progetto.

3.1.4.3.6 Uso dei documenti

I documenti possono essere adibiti ad uso interno o esterno:

- **Interno**: documenti utilizzati all'interno del gruppo, tipicamente *Verbali* e *Norme di Progetto*;
- **Esterno**: documenti destinati a persone esterne, ovvero committenti e proponente.

3.1.4.3.7 Note a piè di pagina

Eventuali note vanno indicate nella pagina corrente, in basso a sinistra. Ogni nota deve riportare un numero e una breve descrizione.

3.1.4.4 Norme tipografiche

Convenzioni sui nomi dei file

I nomi di file (estensione esclusa) e cartelle utilizzano la convenzione "Snake_case$_G$ " e alcune regole aggiuntive elencate di seguito:

1. i nomi dei file composti da più parole usano il carattere *underscore* come carattere separatore, eccetto il file del *Registro delle modifiche* nominato "registroModifiche.tex";
2. i nomi dei file sono scritti interamente in minuscolo;
3. i nomi dei file dei documenti principali, ossia quelli che raggruppano le sezioni, devono contenere anche la versione;
4. le preposizioni **vanno** messe;
5. i nomi delle cartelle seguono la convenzione Snake_case, ma con la prima lettera della prima parola in maiuscolo (e.i. Norme_di_progetto).

Alcuni esempi di file **corretti** sono:

- studio_di_fattibilità_v1.0.0 (contiene la versione);
- processi_di_supporto (file di una sezione delle *Norme di Progetto*, non contiene la versione).

Alcuni esempi di file **non corretti** sono:

- Norme_di_progetto (usa maiuscole);
- norme-di-progetto (non utilizza underscore come separatore);
- norme_progetto (omette la preposizione "di").

Glossario

- ogni termine inserito nel *Glossario* è marcato con una **G** maiuscola a pedice, solamente nella sua prima occorrenza;
- non vengono segnate con la **G** a pedice le parole da *Glossario* presenti nei titoli e nelle didascalie di immagini e tabelle;
- se nel *Glossario* un termine presenta una descrizione che utilizza termini da glossario, è necessario trattare questi termini come tali, segnando la **G** a pedice e aggiungendoli al documento con la relativa descrizione;
- se nel *Glossario* è presente un termine sinonimo (o tradotto il lingua inglese) di un altro già presente, bisognerà collegarlo alla relativa definizione attraverso il comando `\hyperref[par:"nome_paragrafo"]` e la relativa label `\label{par:nome_paragrafo}`, posta sopra la prima occorrenza di definizione.

Stile del testo

- **Grassetto**: viene applicato se necessario alle voci di un elenco puntato, a titoli o a termini di frasi che si vuol far risaltare;
- **Corsivo**: vengono scritti in corsivo e con la prima lettera maiuscola il nome del progetto *Predire in Grafana*, i ruoli, i documenti citati, il nome del gruppo *TeamAFK* ed il nome dell'azienda proponente *Zucchetti SPA*;
- **Maiuscolo**: solo gli acronimi vengono scritti interamente in maiuscolo; nel caso di nomi o titoli composti da più parole verrà indicato con la lettera maiuscola solamente la prima lettera della parola;
- **Nomi dei documenti**:
  - utilizzare il corsivo per citare un documento;
  - ogniqualvolta si cita un documento, bisogna indicare con la lettera maiuscola le iniziali dei nomi di cui è composto, senza indicarne la versione.
    Caso particolare:
    * quest'ultima può essere indicata quando si deve far riferimento ad una **specifica** versione del documento in questione. In questo caso utilizzare la convenzione Snake_case sopra definita: e.i. *Norme_di_Progetto_v1.0.0*.
  - se si utilizza il documento come titolo o in una voce di elenco, si deve seguire la convenzione sopra riportata ma senza utilizzare il corsivo.

Elenchi puntati

Ogni voce di un elenco o sottoelenco comincia per lettera minuscola, e termina per ";", eccetto l'ultima che termina per ".". Se le voci contengono una descrizione, andranno scritte in grassetto e con la prima lettera maiuscola.

Didascalia

Nella didascalia deve comparire il numero della sezione a cui l'immagine o tabella si riferisce, seguita dal numero progressivo delle stesse di quella sezione e una breve descrizione:

X.Y.Z: <descrizione>

- **X.Y**: rappresenta la sezione;
- **Z**: rappresenta il numero progressivo della tabella o immagine nella sezione X.Y;
- **Descrizione**: breve descrizione identificativa.

Formati comuni

In conformità allo standard ISO 8601[1]:

[1]ISO 8601: standard internazionale per la rappresentazione di date e orari.

- le date devono essere scritte secondo il formato gregoriano:
  YYYY-MM-DD
  dove YYYY indica l'anno, MM il mese (da 01 a 12) e DD il giorno (da 01 a 31);
- gli orari devono seguire il formato 24 ore:
  HH:MM
  dove HH indica le ore (da 00 a 23) e MM i minuti (da 00 a 59).

Sigle

Il progetto prevede la redazione di un insieme di documenti, suddivisi in documenti interni ed esterni. Sono di seguito elencati con le rispettive sigle e descrizioni.
I documenti interni sono:

- **Studio di fattibilità - SdF**: descrive in modo sintetico i capitolati e spiega le motivazioni della loro scelta o esclusione;
- **Norme di Progetto - NdP**: sono un riferimento normativo per lo svolgimento del progetto.

I documenti esterni sono:

- **Analisi dei Requisiti - AdR**: stabilisce le caratteristiche che il software deve rispettare;
- **Piano di Progetto - PdP**: descrive la strategia di gestione del progetto, evidenziandone la fattibilità e le criticità;
- **Piano di Qualifica - PdQ**: descrive la qualità del software e dei processi, e come la si intende raggiungere;
- **Glossario - G**: raccoglie i termini di interesse sui quali è necessaria una descrizione più approfondita che ne chiarisca il significato;
- **Manuale Utente - MU**: a disposizione degli utenti;
- **Manuale Sviluppatore - MS**: a disposizione di sviluppatori e manutentori.

I verbali sono un caso particolare di documenti, che possono essere interni o esterni:

- **Verbale - V**: descrivono le interazioni avvenute durante un incontro tra i membri del team (verbale interno) o con il proponente del progetto (verbale esterno).

Le diverse fasi del progetto sono le seguenti:

- **Revisione dei Requisiti - RR**: studio iniziale del capitolato, se ben fatto permette al gruppo di aggiudicarselo;
- **Revisione di Progettazione - RP**: riguarda la definizione dell'architettura del software e di una Proof of Concept$_G$ per mostrarne la fattibilità;
- **Revisione di Qualifica - RQ**: interessa la definizione dettagliata e la codifica del prodotto;

- **Revisione di Accettazione - RA**: se il prodotto soddisfa i requisiti del proponente, viene accettato e rilasciato.

3.1.4.5 Elementi grafici

Immagini

Le immagini sono centrate e hanno una breve didascalia descrittiva sottostante. Tutte le immagini devono aver il formato `.png`$_G$.

Tabelle

Le tabelle sono scritte allo stesso modo in tutti i documenti LaTeX: fare riferimento alla Wiki di Overleaf https://www.overleaf.com/learn/latex/tables.
Ogni tabella deve essere accompagnata dalla propria didascalia descrittiva (caption), da posizionare al di sopra della tabella.
Fanno eccezione le tabelle del *Registro delle Modifiche* che non hanno didascalia.

Diagrammi UML

I diagrammi UML$_G$ vengono inseriti nei documenti sotto forma di immagine.

3.1.5 Metriche di qualità

Le metriche per la qualità dei documenti definiscono l'obiettivo di redarre documenti facilmente leggibili. Esse rispetteranno la seguente notazione:

M[D][numero]

dove:

- D: indica che è una metrica relativa ai documenti;
- numero: identifica in maniera univoca la metrica, assume un valore intero a due cifre incrementale, a partire da 01.

Se nelle formule di calcolo delle metriche è presente il simbolo "#", va inteso come la parola "numero". Un esempio può essere il seguente:

$$\#parole\_doc = numero\ di\ parole\ presenti\ nel\ documento$$

Tabella 3.1.1: Metriche di qualità dei documenti

Nome	Codice	Descrizione
Indice Gulpease (IG)	MD01	È un indice di leggibilità di un testo tarato sulla lingua italiana e basato su due variabili linguistiche: la lunghezza della parola e la lunghezza della frase rispetto al numero delle lettere. La formula per il suo calcolo è: $IG = 89 + \frac{300 \cdot (\#frasi) - 10 \cdot (\#lettere)}{\#parole}$ Il risultato è un valore compreso nell'intervallo tra 0 e 100, dove il valore 100 indica la più alta leggibilità. Un indice inferiore a 80 indica documenti di difficile leggibilità per chi ha la licenza elementare, inferiore a 60 per chi ha la licenza media, inferiore a 40 per chi ha un diploma superiore.
Indice Fog (IF)	MD02	Misura la lunghezza media delle parole e delle frasi nei documenti. Quanto più è alto il valore dell'indice Fog, tanto più è difficile capire il documento. La formula per il suo calcolo è: $IF = 0.4 \cdot [(\frac{\#parole}{\#frasi}) + 100 \cdot (\frac{\#parole\_complesse}{\#parole})]$ Il numero risultante è un indicatore del numero di anni di educazione formale della quale una persona necessita al fine di leggere il testo con facilità.

3.1.6 Strumenti di supporto

3.1.6.1 LaTeX

LaTeX è lo strumento scelto per la stesura dei documenti, un linguaggio basato sul programma di composizione tipografica TeX, che permette di scrivere documenti in modo ordinato, modulare, collaborativo e scalabile.

3.1.6.2 TeXmaker

TeXmaker è l'editor utilizzato per la stesura del codice LaTeX. Questo strumento, oltre ad integrare un compilatore e un visualizzatore PDF, fornisce suggerimenti di completamento per comandi LaTeX.

https://www.xm1math.net/texmaker/

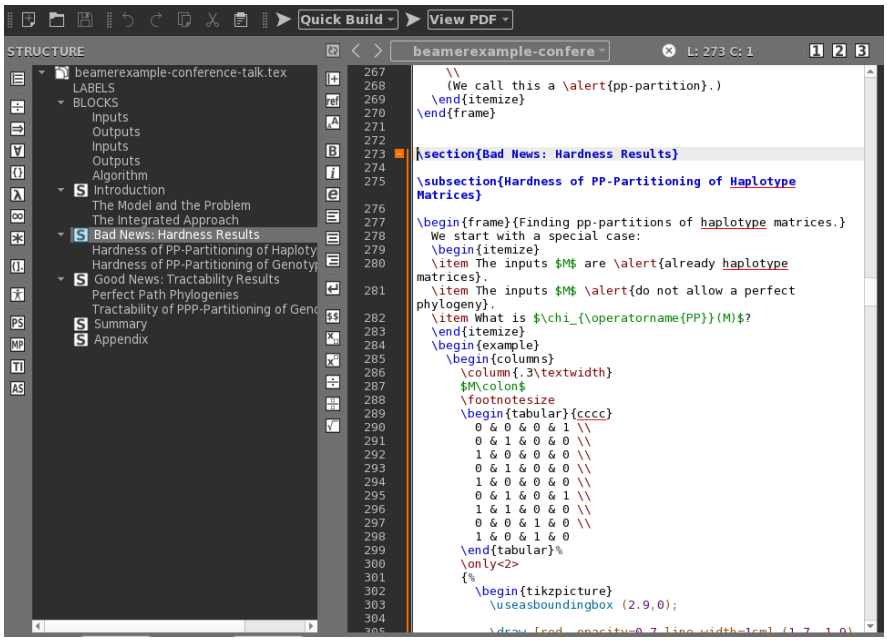

Figura 3.1.1: TEXmaker - per la stesura dei documenti

3.1.6.3 GanttProject

GanttProject è un programma gratuito dedicato alla produzione dei diagrammi di Gantt$_G$. Permette di creare task e milestone$_G$, organizzare le task in lavoro strutturato a interruzioni, disegnare i vincoli di dipendenza tra di esse e molte altre utilità, generando automaticamente il relativo diagramma.

https://www.ganttproject.biz/

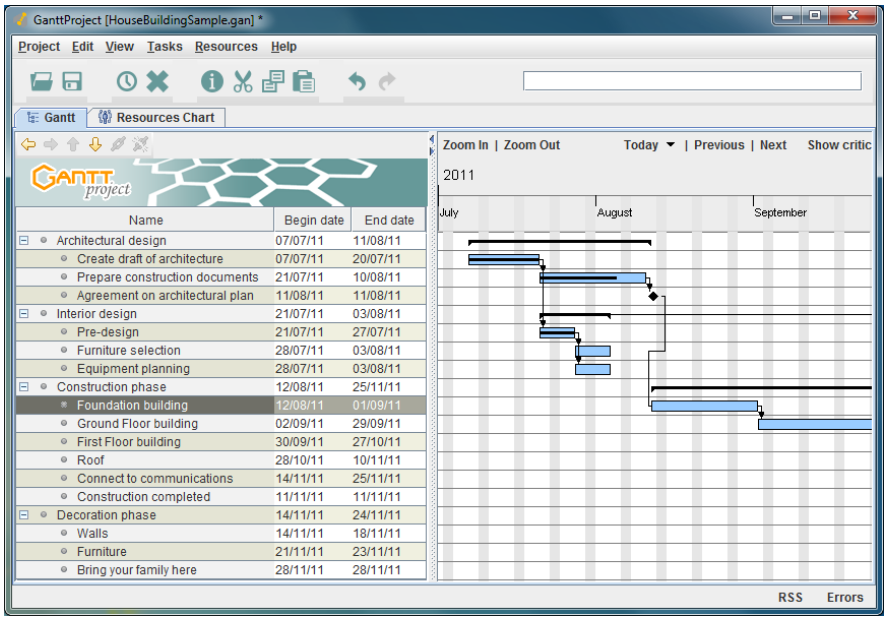

Figura 3.1.2: GanttProject - per la relizzazione di diagrammi di Gantt

3.1.6.4 Draw.io

Draw.io viene utilizzato per la produzione degli UML.
https://www.draw.io/

3.2 Gestione della configurazione

3.2.1 Scopo

L'obiettivo della configurazione è di creare ordine tra i documenti e il software. Tutto ciò che è configurato ha uno stato identificativo, è modificato secondo regole ben definite ed è posto sotto versionamento$_G$.

3.2.2 Aspettative

Questa sezione ha lo scopo supportare i processi 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. Viene quindi descritta la configurazione del repository su GitHub, del sistema di versionamento Git e dei servizi associati a GitHub.

3.2.4 Attività

3.2.4.1 Versionamento

Ogni versione di qualsiasi documento deve corrispondere ad una riga della tabella delle modifiche. Il numero di versione è composto da tre cifre:

X.Y.Z

- **X**: rappresenta una versione stabile del documento, resa tale dopo l'approvazione del *Responsabile* di progetto:
  - inizia da 0;
  - viene incrementata di un'unità alla volta.
- **Y**: indica l'ultima versione del documento che ha passato la fase di verifica:
  - inizia da 0;
  - viene incrementato dal verificatore ad ogni verifica;
  - quando viene incrementato X, viene riportato a 0.
- **Z**: indica l'ultima modifica apportata al documento dal redattore:
  - inizia da 0;

- viene incrementato dal redattore del documento ad ogni modifica;
- quando viene incrementato Y, viene riportato a 0.

Apprese le correzioni effettuate dal docente, il *TeamAFK* ha deciso di modificare il significato che questi parametri assumevano, consentendo di ottenere uno scatto di versione solo successivamente ad un'operazione di verifica. Il significato che i nuovi parametri assumono sono:

- **X**: un suo aumento determina che il documento sia stato modificato, verificato e approvato, quindi ritenuto pronto al rilascio;
- **Y**: un suo aumento determina la conclusione di un'attività di modifica e di verifica di un processo ritenuto di entità maggiore, come può essere l'inserimento di una nuova sezione o una sua modifica sostanziale (refactoring);
- **Z**: un suo aumento determina la conclusione di un'attività di modifica e di verifica di un processo ritenuto di entità minore, come può essere una correzione, un'aggiunta minore o un aggiornamento di una specifica sezione.

L'aumento del parametro X determina l'azzeramento dei parametri Y e Z.

3.2.4.2 GitHub

Per le parti del progetto da versionare si è scelto di usare GitHub$_G$, un servizio del sistema di versionamento distribuito Git per contenere la repository remota.
I membri del team possono interagire con il VCS$_G$ sia da linea di comando, sia attraverso software che ne migliorano l'usabilità, come GitKraken e GitHub Desktop. La versione ufficiale del progetto è ospitata in una repository remota su GitHub, all'indirizzo

https://github.com/teamafkSWE

3.2.4.2.1 Struttura del repository

All'interno della repository principale sopra descritta troviamo due differenti repository:

- **PredireInGrafana-docs**: contiene tutti i documenti ufficiali del progetto, suddivisi in specifiche cartelle:
  https://github.com/teamafkSWE/PredireInGrafana-docs
  - **Cartella principale - Presentazioni**: raccoglie le presentazioni relative ad ogni consegna;
  - **Cartella principale - RR**: raccoglie i file sorgenti per la compilazione dei documenti, suddivisi tra esterni ed interni, realizzati per la *Revisione dei Requisiti*;
  - **Cartella principale - RP**: raccoglie i file sorgenti per la compilazione dei documenti, suddivisi tra esterni ed interni, realizzati per la *Revisione di Progettazione*. In futuro saranno aggiunte cartelle distinte nominate **RQ** e **RA**, contenenti i file delle rispettive consegne.

  - **Cartella principale - RQ**: raccoglie i file sorgenti per la compilazione dei documenti, suddivisi tra esterni ed interni, realizzati per la *Revisione di Qualifica*. In futuro sarà aggiunta la cartella relativa alla **RA**, contenenti i file della rispettiva consegna.
    - **Tipologia_di_documento**: ogni documento avrà la rispettiva cartella (e.i. Norme_di_progetto), contenente tutti i file (sezioni ed immagini) necessari per la sua compilazione;
    - **copertina.tex**: file che permetterà una facile e rapida modifica dell'impostazione testuale del frontespizio.
  - **Template**: contiene tutti i file che definiscono il template LaTeX per la creazione di nuovi documenti.
- **PredireInGrafana-SW**: conterrà tutti i file di codifica dei plug-in da sviluppare.
  https://github.com/teamafkSWE/PredireInGrafana-SW

Entrambe le repository, avranno una propria struttura identica a livello:

- **Locale**: ogni membro del gruppo lavora sui file clonati dal repository remoto nel proprio PC;
- **Remoto**: presente su GitHub, contiene il lavoro svolto da ogni componente e che viene condiviso con il team.

3.2.4.2.2 Tipi di file

I file utilizzati per la documentazione del progetto sono:

- file con estensione .tex di LaTeX;
- file con estensione .pdf (da consegnare);
- file di stile, .sty, e immagini di supporto.

Il file ".gitignore" è presente al livello più esterno della repository ed elenca tutti i file esclusi dal versionamento.

3.2.4.2.3 Utilizzo di Git

Il repository di Git è composto da vari branch$_G$, per favorire la collaborazione tra i vari membri e il parallelismo delle attività. Si consiglia quindi di seguire questa procedura:

1. scegliere il proprio branch di lavoro;
2. eseguire il pull$_G$ dal repository remoto, che effettua quindi l'aggiornamento del proprio repository locale;
3. svolgere il compito assegnato;
4. eseguire il comando di aggiunta (add) dei file nuovi o modificati da condividere all'area di staging$_G$;

5. eseguire il comando di commit$_G$ dei file aggiunti, corredato da un messaggio che identifica il lavoro svolto;
6. eseguire il push$_G$ del commit sul repository remoto.

3.2.4.3 Gestione delle modifiche

Tutti i membri del team possono modificare i file in ogni branch, ad eccezione del branch master, per il quale occorre richiedere una pull e ottenere l'approvazione di un altro membro. Per effettuare modifiche maggiori sui contenuti o sulla struttura dei file è previsto:

- contattare il *Responsabile* del file da modificare;
- suggerire la modifica da effettuare;
- se il responsabile valuta positivamente la modifica, allora la applica.

Per modifiche minori, come correzioni grammaticali o miglioramenti sintattici, si consiglia di modificare indipendentemente. In entrambi i casi è opportuno commentare i propri commit con chiarezza per risalire facilmente alle modifiche effettuate.

3.2.5 Metriche di qualità

Per questo processo non sono state definite metriche di qualità.

3.2.6 Strumenti di supporto

3.2.6.1 GitHub

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*.
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 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 Coveralls per eseguire le verifiche automatiche sul codice sorgente.

3.2.6.2 Coveralls

Servizio web utilizzato per tenere traccia della copertura del codice.
https://coveralls.io/

3.3 Gestione della qualità

3.3.1 Scopo

Lo scopo è garantire che il prodotto e i servizi offerti rispettino gli obiettivi di qualità e che i bisogni del proponente siano soddisfatti.

3.3.2 Aspettative

Le aspettative di questo processo 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 sviluppato;
- oggettività nella ricerca della qualità in modo da ottenerla in tutti i processi e in tutti i prodotti;
- qualità misurabile attraverso delle metriche.

3.3.3 Descrizione

La gestione della qualità viene approfondita nel *Piano di Qualifica*, dove sono descritte le modalità utilizzate per garantire la qualità nello sviluppo del progetto. In particolare:

- sono presentati gli standard utilizzati;
- sono individuati i processi$_G$ di interesse;
- sono individuati gli attributi del software più importanti per il progetto.

Per ogni processo vengono descritti:

- gli obiettivi da perseguire;
- le strategie da applicare;
- le metriche da utilizzare.

L'obiettivo è ottenere software e documentazione di qualità soddisfacente.

3.3.4 Attività

3.3.4.1 Classificazione delle metriche

Le metriche sono distinte in tre categorie: processi, documentazione e codifica. Per ciascuna di esse si indica il motivo per cui è stata scelta e la procedura di calcolo.
Le metriche rispetteranno la seguente notazione:

M[categoria][numero]

dove:

- Categoria: indica la categoria della metrica, più precisamente:

  - **G** per indicare le metriche riferite al prodotto, quindi a carattere più generale;
  - **P** per indicare le metriche dei processi;
  - **D** per indicare le metriche dei documenti;
  - **S** per indicare le metriche della codifica del software e dei relativi test.
- numero: identifica in maniera univoca la metrica in ogni categoria, assume un valore intero a due cifre incrementale, a partire da 01.

Se nelle formule di calcolo delle metriche è presente il simbolo "#", va inteso come la parola "numero". Un esempio può essere il seguente:

$$\#parole\_doc = numero\ di\ parole\ presenti\ nel\ documento$$

3.3.5 Metriche di qualità

Tabella 3.3.1: Metriche per la qualità generale del prodotto

Nome	Codice	Descrizione
Percentuale di Metriche Soddisfatte (PROC)	MG01	Indica la percentuale di metriche soddisfatte rispetto alla totalità di metriche da utilizzare all'interno del progetto. Si calcola con la seguente formula: $$PROC = \frac{metriche\_soddisfatte}{metriche\_totali} \cdot 100$$
Percentuale di Requisiti Obbligatori Soddisfatti (PROS)	MG02	Indica la percentuale di requisiti obbligatori soddisfatti rispetto alla totalità di requisiti obbligatori da sviluppare e rispettare. Si calcola con la seguente formula: $$PROS = \frac{requisiti\_obbligatori\_soddisfatti}{requisiti\_obbligatori\_totali} \cdot 100$$
Percentuale di Requisiti Desiderabili Soddisfatti (PRDS)	MG03	Indica la percentuale di requisiti desiderabili soddisfatti rispetto alla totalità di requisiti desiderabili da sviluppare e rispettare. Si calcola con la seguente formula: $$PROS = \frac{requisiti\_desiderabili\_soddisfatti}{requisiti\_desiderabili\_totali} \cdot 100$$

3.3.6 Strumenti di supporto

Gli strumenti di supporto utilizzati per garantire la qualità sono le metriche di qualità.

3.4 Verifica

3.4.1 Scopo

Il processo di verifica ha come scopo la realizzazione di prodotti corretti, coesi e completi. Sono soggetti a verifica i documenti e il software. Il processo di verifica deve rispettare i seguenti punti:

- la verifica deve essere effettuata seguendo procedure ben definite;
- per verificare vi sono criteri chiari e affidabili;
- ogni fase del prodotto viene verificata;
- dopo la verifica il prodotto è in uno stato stabile;
- il prodotto può essere validato.

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 del prodotto;
- avere a disposizione dei criteri e procedure, comprensibili e affidabili, da seguire.

3.4.3 Descrizione

Il processo di verifica assicura che tutte le attività svolte della fase in esame, attraverso analisi e test successivamente definiti, siano conformi alle aspettative; prende in input ciò che è già stato prodotto e lo restituisce in uno stato stabile.
Per ottenere tale risultato ci si affida a processi di analisi e test.

3.4.4 Attività

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

3.4.4.1 Analisi statica

L'analisi statica effettua controlli su documenti e codice. Questo tipo di analisi serve per rilevare varie tipologie di anomalie.
Questa attività viene effettuata con l'ausilio di due metodi manuali di lettura (attuati da persone) differenti:

- **Walkthrough**: i vari componenti del team effettuano una lettura scrupolosa di documenti e/o codice alla ricerca di errori, senza sapere inizialmente se ce ne siano;
- **Inspection**: i verificatori usano liste di controllo (checklist) per fare ispezione cercando errori specifici in parti specifiche. Questa tecnica permette quindi di aumentare l'efficienza dello sviluppo, diminuendo i tempi dell'analisi statica.

A seguire sono descritte le liste di controllo utilizzabili per le ispezioni:

Tabella 3.4.1: Errori frequenti nei documenti.

Oggetto	Controllo
Formato data	Deve seguire il formato gregoriano YYYY-MM-DD.
Sintassi	La frase è troppo complessa e deve essere semplificata.
Punteggiatura degli elenchi	Ogni voce termina in ";" eccetto l'ultima che termina in ".".
Errori di battitura	Tipici errori di battitura dovuti alla vicinanza delle lettere sulla tastiera, ad esempio la lettera "a" al posto della "s", "i" al posto di "o", "m" al posto di "n".
Uso degli accenti	Utilizzare le lettere accentate presenti nella tastiera o utilizzare il codice ALT.
Inconsistenze nell'uso delle iniziali maiuscole nei titoli delle parti di documento	Ogni titolo deve avere la prima lettera della prima parola maiuscola e il resto minuscolo, tranne quando ci si riferisci ad attività o documenti specifici.
Gerarchia delle sezioni	Non viene scritta la gerarchia delle sezioni dei documenti in maniera adeguata, ovvero: section, subsection, subsubsection, paragraph, subparagraph, subsubparagraph.
Uso corretto del comando `\glo`	Il comando `\glo` è stato creato per le parole da inserire nel glossario; va indicato solamente nella prima occorrenza della parola da inserire nel glossario.

3.4.4.2 Analisi dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che richiede la sua esecuzione. Produce una misura della qualità del prodotto, mediante l'esecuzione di test specifici che verificano se il prodotto funziona e se ci sono anomalie.

3.4.4.2.1 Test

I test sono l'attività fondamentale dell'analisi dinamica: il loro scopo è verificare che il codice scritto funzioni correttamente. I test devono:

- essere ripetibili;
- specificare l'ambiente di esecuzione;
- identificare input e output richiesti;
- avvertire di possibili effetti indesiderati;
- fornire informazioni sui risultati dell'esecuzione.

Ci sono vari tipi di test, ognuno dei quali ha un diverso oggetto di verifica e scopo.

Test funzionali

Test condotti per valutare la conformità di un componente o sistema con requisiti$_G$ funzionali. Ne fanno parte:

- **Test di unità**: si eseguono su unità di software, e si concentrano sul loro funzionamento individuale;
- **Test di integrazione**: verificano se sono rispettati i contratti di interfaccia tra più moduli o sub-system (interni o esterni);
- **Test di accettazione**: gli UAT (User Acceptance Testing) si occupano di verificare il prodotto e, in particolare, il soddisfacimento del cliente. Il superamento di questi test garantisce che il software sia pronto per essere rilasciato.

Test di regressione

Il test di regressione va eseguito ogni volta che viene modificata un'implementazione in un programma. È possibile eseguire nuovamente i test esistenti sul codice modificato, integrando solo le parti che abbiano precedentemente superato il test di unità, per stabilire se le modifiche apportate hanno alterato elementi precedentemente funzionanti.
Se necessario è anche possibile scrivere nuovi test.

Test di sistema

Verificano il comportamento dell'intero sistema.
Lo scopo principale di questi test è la verifica del sistema rispetto alle specifiche tecniche definite nel'*Analisi dei Requisiti*.

Codifica dei test

La codifica dei test è la seguente:

TX[Tipo_Requisito][Importanza][CodiceUC]

dove:

- **X**: indica il tipo di test. Può assumere i seguenti valori:
  - **U**: indica un test di unità;

  - **I**: indica un test di integrazione;
  - **A**: indica un test di accettazione;
  - **R**: indica un test di regressione;
  - **S**: indica un test di sistema.
- **Tipo_Requisito**: indica il tipo del requisito. Esso può essere:
  - **O**: obbligatorio;
  - **D**: desiderabile;
  - **F**: facoltativo.
- **Importanza**: indica l'importanza del requisito. Viene indicata con:
  - **F**: per indicare un requisito funzionale;
  - **V**: per indicare un requisito di vincolo;
  - **Q**: per indicare un requisito di qualità;
  - **P**: per indicare un requisito prestazionale.
- **CodiceUC**: rappresenta il codice identificativo crescente del componente da verificare.

3.4.5 Metriche di qualità

Le metriche descritte in questa sezione sono metriche software utilizzate per verificare la qualità e la completezza dei test effettuati. Più in dettaglio:

Tabella 3.4.2: Metriche per la qualità e completezza dei test

Nome	Codice	Descrizione
Code Coverage (CC)	MS07	Percentuale delle linee di codice coperte dai test. Avere codice coperto da test riduce la possibilità di introdurre errori nel prodotto. $$CC[\%] = \frac{\#righe\_codice\_testate}{\#tot\_righe\_codice}$$
Passed Test Cases Percentage (PTCP)	MS08	Percentuale di test superati sul totale dei test eseguiti. Un'alta percentuale di superamento indica la tendenza del codice ad essere già corretto prima dei test. Si calcola con la seguente formula: $$PTCP[\%] = \frac{\#test\_superati}{\#test\_eseguiti} \cdot 100$$

Tabella 3.4.2: (continua)

Nome	Codice	Descrizione
Failed Test Cases Percentage (FTCP)	MS09	Percentuale di test falliti sul totale dei test eseguiti. Un'alta percentuale di fallimento indica la tendenza alla scorrettezza del codice. Si calcola con la seguente formula: $$FTCP[\%] = \frac{\#test\_falliti}{\#test\_eseguiti} \cdot 100$$

3.4.6 Strumenti di supporto

3.4.6.1 Correzione ortografica

Il controllo ortografico viene eseguito principalmente basandosi sugli strumenti integrati in TEXmaker, il quale fornisce un dizionario italiano e sottolinea in rosso le parole che non vi appartengono.

3.4.6.2 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 di validazione è accertare che il prodotto finale corrisponda alle attese, soddisfando tutti i requisiti concordati e i bisogni del committente.

3.5.2 Aspettative

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

3.5.3 Descrizione

Le attese sono quelle del committente e corrispondono all'*analisi_dei_requisiti_v3.0.0*. Per eseguire una validazione si deve avere un prodotto ad un sufficiente livello di avanzamento, quindi il numero di validazioni svolte è minore rispetto al numero delle verifiche. Durante questo processo vengono svolti i test di accettazione tramite i quali il proponente valida il prodotto software.

3.5.4 Attività

3.5.4.1 Test di accettazione

Gli UAT si occupano di verificare il prodotto e, in particolare, il soddisfacimento del cliente. Il superamento di questi test garantisce che il software sia pronto per essere rilasciato.

3.5.5 Metriche di qualità

Per questo processo non sono state definite particolari metriche di qualità.

3.5.6 Strumenti di supporto

Per questo processo non vengono utilizzati degli strumenti di supporto.

3.6 Gestione dei cambiamenti

3.6.1 Scopo

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

3.6.2 Aspettative

Questo approccio porta i seguenti vantaggi, fondamentali per rispettare il *Piano di Progetto*:

- rispetto degli obiettivi prefissati;
- rispetto dei tempi previsti;
- rispetto del budget preventivato.

3.6.3 Descrizione

Il Change Management è un approccio strutturato al cambiamento negli individui, nei gruppi e nelle organizzazioni che rende possibile la transizione da un assetto corrente a un futuro assetto desiderato. Un programma strutturato di sviluppo e crescita delle risorse umane è un elemento fondamentale di accompagnamento al cambiamento.
Pertanto, il *TeamAFK* ha scelto di seguire l'approccio **agile** per la gestione del progetto, in cui i requisiti e le soluzioni maturano in corso d'opera attraverso la collaborazione fra stakeholders. Si tratta quindi di produrre il più rapidamente possibile i prototipi e poi rifinirli attraverso cicli successivi di miglioramento.

3.6.4 Attività

3.6.4.1 Adattamento ed implementazione del processo

Per avvalorare quanto detto prima ed implementare tale processo al progetto in corso, il *TeamAFK* ha deciso di adottare il modello 4P:

- **People**: il Change Management efficace mette l'utente al centro. Il *Responsabile del Progetto* gestisce le comunicazioni interne tra i membri del gruppo e quelle esterne con il proponente e/o committente;
- **Process**: i processi vengono rivisti in chiave moderna ed efficace; devono essere circolari e chiusi, 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.
- **Platform**: il *TeamAFK* utilizza tecnologie avanzate a supporto dello sviluppo del progetto (ESLint, Coveralls, TravisCI);
- **Place**: si intendono i luoghi di lavoro dove viene svolto il progetto. Il *TeamAFK* si vede obbligato in questo senso ad attuare lo Smart Working, a causa delle limitazioni dovute da Covid19.

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 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, problemi compresi, sono rilevati in un prodotto software o in un'attività, deve essere realizzato un report che li descriva. Questo report deve essere usato come componente del punto precedente: dall'individuazione dei cambiamenti al processo di analisi e risoluzione, fino all'individuazione della causa ed alla sua analisi per individuarne la natura. 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.

3.6.5 Metriche di qualità

Per questo processo non sono state definite delle metriche di qualità specifiche.

3.6.6 Strumenti di supporto

Per questo processo non vengono utilizzati degli strumenti di supporto.

4 Processi organizzativi

4.1 Gestione organizzativa del progetto

4.1.1 Scopo

Lo scopo del processo 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* e redatte all'interno del documento *piano_di_progetto_v3.0.0*.

4.1.2 Aspettative

Gli obiettivi del processo di gestione organizzativa del progetto 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 e i prodotti in modo efficace.

4.1.3 Descrizione

Il processo di gestione organizzativa del progetto è composto dalle seguenti attività:

- definizione dello scopo;
- stima e pianificazione di risorse, costi e tempo per lo svolgimento del progetto;
- 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 Attività

4.1.4.1 Pianificazione delle risorse

Per gestire le risorse disponibili per il nostro progetto dobbiamo monitorare il rispetto dei costi e dei tempi preventivati. A tal proposito definiamo i ruoli di progetto e le relative funzioni.

4.1.4.2 Ruoli di progetto

Ciascun membro del gruppo, a rotazione, deve ricoprire il ruolo che gli viene assegnato e che corrisponde all'omonima figura aziendale. Nel *Piano di Progetto* vengono organizzate e pianificate le attività assegnate agli specifici ruoli. I ruoli che ogni componente del gruppo è tenuto a rappresentare sono descritti di seguito.

Assegnazione I ruoli scelti corrispondono alla rispettiva figura aziendale e valgono per la durata di una milestone (consegna). 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.

Responsabile di progetto

Il *Responsabile di Progetto* (o semplicemente *Responsabile*) è una figura chiave in quanto ricadono su di lui le responsabilità di pianificazione, gestione, controllo e coordinamento delle risorse e attività del gruppo. Il *Responsabile* si occupa anche di interfacciare il gruppo con le persone esterne facendo da intermediario: sono quindi di sua competenza le comunicazioni con committente e proponente. Questa figura è incaricata anche di analizzare e gestire le criticità, che si incontrano durante il progetto, e di approvare i documenti.

Amministratore

L'*Amministratore* ha il compito di supporto e controllo dell'ambiente di lavoro. Egli deve quindi:

- dirigere le infrastrutture di supporto;
- risolvere problemi legati alla gestione dei processi;
- gestire la documentazione;
- controllare versioni e configurazioni.

Analista

L'*Analista* si occupa di analisi dei problemi e del dominio applicativo. Questa figura ha anche il compito di redigere i documenti, in questo caso può essere definito come *Redattore*. Le sue responsabilità sono:

- studio del dominio del problema;
- definizione della complessità e dei requisiti dello stesso;
- redazione del documenti: *Analisi dei Requisiti* e *Studio di Fattibilità*.

Progettista

Il *Progettista* gestisce gli aspetti tecnologici e tecnici del progetto. Il *Progettista* deve:

- effettuare scelte efficienti ed ottimizzate su aspetti tecnici del progetto;
- sviluppare un'architettura che sfrutti tecnologie note ed ottimizzate, su cui basare un prodotto stabile e manutenibile.

Programmatore

Il *Programmatore* è responsabile della codifica del progetto e delle componenti di supporto che serviranno per effettuare le prove di verifica e validazione sul prodotto. Il *Programmatore* si occupa di:

- implementare le decisioni del progettista;
- creare o gestire componenti di supporto per la verifica e validazione del codice.

Verificatore

Il *Verificatore* si occupa di controllare il prodotto del lavoro svolto dagli altri membri del team, sia esso codice o documentazione. Per le correzioni si affida agli standard definiti nelle *Norme di Progetto*, nonché alla propria esperienza e capacità di giudizio. Il *Verificatore* deve:

- ispezionare i prodotti in fase di revisione, avvalendosi delle tecniche e degli strumenti definiti nelle *Norme di Progetto*;
- evidenziare difetti ed errori del prodotto in esame;
- segnalare eventuali errori trovati all'autore dell'oggetto preso in esame o alla persona che ha responsabilità su di esso.

4.1.4.3 Gestione dei rischi

È compito del *Responsabile* rilevare i rischi e renderli noti, tramite un continuo monitoraggio e una continua identificazione di quest'ultimi.
Qualora dovesse essere **identificato** un nuovo rischio è necessario procedere con i seguenti passaggi:

1. **Classificare** il rischio seguendo la codifica;
2. **Descrivere** una strategia da applicare per gestire il rischio;
3. **Riportare** il rischio nel *Piano di Progetto*.

4.1.4.3.1 Codifica

La codifica dei rischi è utilizzata per la classificazione. Il codice di un rischio si presenta nella forma:

Ri[Categoria][Numero]

dove:

- **Categoria**: indica la categoria del rischio, essa può assumere i valori:
  - **O** per i rischi organizzativi;
  - **T** per i rischi tecnologici;
  - **P** per i rischi interpersonali.
- **Numero**: insieme a categoria identifica in maniera univoca il rischio, può assumere un valore intero progressivo a due cifre (01-99).

4.1.4.4 Gestione delle comunicazioni

4.1.4.4.1 Comunicazioni interne

Le comunicazioni interne ai membri del gruppo vengono gestite tramite 2 applicazioni:

- Telegram$_G$;
- Discord$_G$.

È stato disposto un gruppo Telegram sul quale si discute di tematiche generali o collettive, garantendo risposte rapide e ordinate in caso di decisioni per votazione, grazie ad apposite funzioni dette bot di Telegram$_G$.
Discord viene usato principalmente per le riunioni tra i membri del gruppo, ma l'applicazione mette anche a disposizione dei canali testuali. Tali canali sono stati suddivisi per tema e vengono usati per le comunicazioni specifiche per agevolare la stesura dei documenti. Vengono suddivisi in:

- **General**: per discussioni riguardanti rotazioni dei ruoli e decisione degli argomenti da discutere nelle riunioni;
- **Links**: per tenere traccia di tutti i link utili al progetto;
- **Analisi-requisiti**: per discutere gli Use Case$_G$ e i requisiti necessari alla stesura dell'*Analisi dei Requisiti*;
- **Norme**: per discutere riguardo le regole del *Way of Working* del gruppo, le norme da seguire e, di conseguenza, la stesura del documento *Norme di Progetto*$_G$;
- **Piano-progetto**: per confrontarsi riguardo il monte ore dei vari ruoli e per facilitare la stesura del documento *Piano di Progetto*;
- **Piano-qualifica**: per discutere di strategie da attuare per garantire qualità attraverso verifica$_G$ e validazione$_G$.

4.1.4.4.2 Comunicazioni esterne

Le comunicazioni con soggetti esterni al gruppo sono di competenza del responsabile. Gli

strumenti predefiniti sono la posta elettronica, dove viene utilizzato l'indirizzo gruppoafk15@gmail.com. Per comunicare con *Zucchetti SPA* viene usato il servizio Skype$_G$ per le chiamate. Il responsabile ha il compito di tenere informati gli altri componenti del gruppo in caso di assenza.

4.1.4.4.3 Gestione riunioni

Le riunioni possono essere interne o esterne. All'inizio di ogni riunione il *Responsabile* nomina, tra i componenti del gruppo, un *Segretario* che si occuperà di far rispettare l'ordine del giorno. Inoltre ha l'onere della stesura del *Verbale di Riunione*$_G$.

Riunioni interne

È compito del *Responsabile* organizzare riunioni interne al gruppo. Ciò prevede, più nello specifico, la stesura dell'ordine del giorno e stabilire data, orario e luogo di incontro, mediando se necessario con i membri per permettere la presenza di tutti. Le riunioni sono tenute principalmente usando Discord, così da essere il più facilmente raggiungibili. Il *Responsabile* deve inoltre assicurarsi, attraverso la comunicazione mediante i mezzi propri del gruppo, che ogni componente sia pienamente a conoscenza della riunione in tutti i suoi dettagli.
D'altro canto ogni membro del gruppo deve presentarsi puntuale agli appuntamenti, e comunicare in anticipo eventuali ritardi o assenze adeguatamente giustificate.

Riunioni esterne

È nuovamente compito del *Responsabile* organizzare riunioni esterne. Nello specifico egli deve preoccuparsi di contattare l'azienda proponente per fissare gli incontri e qualora sia necessario, tenendo conto anche delle preferenze di date e orario espresse dagli altri membri del gruppo. La partecipazione a tali riunioni deve essere, a meno di casi eccezionali, unanime. Ogni membro del gruppo può, inoltre, esprimere al Responsabile una richiesta, adeguatamente motivata, di fissare una riunione esterna. A questo punto sarà compito dello stesso Responsabile giudicare come valida o meno la richiesta presentatagli ed agire di conseguenza.

Verbale di riunione

Ad ogni riunione, interna o esterna, è compito del *Segretario* designato redigere il *Verbale di riunione* corrispondente, che deve essere poi approvato dal *Responsabile*. La struttura del *Verbale* è definita in § 3.1.4.3.5.

4.1.5 Metriche di qualità

L'obiettivo è pianificare le risorse per garantire un corretto avanzamento del progetto, monitorando e rispettando costi e tempistiche.

Tabella 4.1.1: Metriche di qualità per la pianificazione efficiente delle risorse

Nome	Codice	Descrizione
Schedule Variance (SV)	MP01	La Schedule Variance indica se una certa attività o processo è in anticipo, in pari, o in ritardo rispetto alla data di scadenza prevista. È calcolata utilizzando la seguente formula: $$SV = DCE - DCP$$ dove: • **DCE**: data conclusione effettiva; • **DCP**: data conclusione pianificata. Se $SV \leq 0$ significa che l'attività o il processo è in pari o in anticipo, invece, se $SV > 0$ significa che l'attività è in ritardo.
Budget Variance (BV)	MP02	Permette di controllare i costi sostenuti alla data corrente rispetto al budget preventivato. Viene calcolata in fase di consuntivo di periodo utilizzando la seguente formula: $$BV[\%] = \frac{CP - CE}{CE} \cdot 100$$ dove: • **CP**: costo preventivato; • **CE**: costo effettivo. Se $BV[\%] \geq 0$ indica che il budget sta venendo speso più lentamente di quanto pianificato, se negativo invece indica che il budget sta venendo speso più velocemente di quanto pianificato.
Produttività (P)	MP03	Rappresenta la produttività media delle risorse impiegate, cioè delle persone coinvolte, nelle diverse fasi del progetto. È misurata in termini di numero di linee di codice (LOC) sviluppate da una persona nell'unità di tempo stabilita (settimana). È utilizzata per valutare lo sforzo richiesto per lo sviluppo del progetto a fronte delle sue dimensioni. $$Pmedia = \frac{LOC}{settimana}$$

4.1.6 Strumenti di supporto

Il gruppo, nel corso del progetto, ha utilizzato o utilizzerà i seguenti strumenti:

- **Telegram**: strumento di messaggistica usato per comunicazioni veloci tra i membri;
- **Discord**: per comunicazioni specifiche o per riunioni interne;
- **Git**: sistema di controllo di versionamento;
- **Gitflow**: sistema per agevolare varie operazioni su Git;
- **GitHub**: per il versionamento e il salvataggio in remoto di tutti i file riguardanti il progetto;
- **GanttProject**: software OpenSource$_G$ usato per la realizzazione dei diagrammi di Gantt;
- **Google Doc**: editor di testo cloud, usato per tenere degli appunti modificabili da tutti;
- **Google Drive**: utilizzato per il salvataggio in remoto dei file non sottoposti a versionamento, in modo da essere reperibili a tutti i membri;
- **Google Calendar**: per tenere traccia delle varie scadenze o riunioni fissate;
- **Skype**: servizio che offre possibilità di fare videoconferenze e chiamate VoIP, utilizzato per comunicare con il proponente;
- **Sistema Operativo**: i requisiti non indicano la necessità di usare un sistema operativo specifico, verranno quindi utilizzati Windows, Linux e Mac OS dai diversi membri del team.

4.2 Formazione dei membri del gruppo

4.2.1 Scopo

Lo scopo del processo di formazione è garantire che ogni membro del gruppo abbia conoscenze e capacità sufficienti per svolgere le attività assegnategli. Per una migliore organizzazione, il lavoro sarà suddiviso in compiti (task).

4.2.2 Aspettative

L'aspettativa del gruppo, nel caso di questo processo, 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 di formazione è composto da un insieme di attività (task) 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*. Al termine dello studio, egli deve comunicare ciò che ha imparato agli altri membri per garantire che tutti apprendano le nozioni. Qualora un componente del gruppo ritenga di non essere in

grado di svolgere un task, dovrà segnalarlo immediatamente al *Responsabile di Progetto* che dovrà organizzare le attività necessarie all'apprendimento.

4.2.4 Attività

4.2.4.1 Guide e documentazione

I membri del gruppo *TeamAFK* provvedono in modo autonomo allo studio individuale delle tecnologie che verranno utilizzate nel corso del progetto. Si vuole operare secondo il principio del miglioramento continuo. Il versionamento dei prodotti servirà anche per apprendere dall'operato altrui, in modo da integrare le conoscenze personali migliorando la qualità e l'efficienza delle attività.

4.2.4.2 Condivisione del materiale

Ogni componente del gruppo è libero di utilizzare per l'apprendimento personale altro materiale oltre a quello indicato in questo documento. Nel caso in cui lo ritenesse utile, potrà inoltre condividere tale materiale, utilizzando il canale comunicativo principale del gruppo (Telegram).

4.2.5 Metriche di qualità

Per questo processo non sono state definite metriche di qualità specifiche.

4.2.6 Strumenti di supporto

Per questo processo non vengono utilizzati degli strumenti di supporto.

A Standard di qualità

A.1 ISO/IEC 9126

Con la sigla ISO/IEC 9126 si individua una serie di normative e linee guida preposte a descrivere un modello di qualità del software.
La norma tecnica relativa alla qualità del software si compone in quattro parti:

- modello della qualità del software;
- metriche per la qualità esterna;
- metriche per la qualità interna;
- metriche per la qualità in uso.

A.1.1 Modello della qualità del software

Il modello di qualità è classificato da sei caratteristiche generali e da varie sottocaratteristiche misurabili attraverso delle metriche.
Di seguito vengono definite tali caratteristiche.

A.1.1.1 Funzionalità

È la capacità del software di soddisfare determinate esigenze, stabilite nell' *Analisi dei Requisiti*, necessarie per operare sotto condizioni specifiche.
In dettaglio il software deve soddisfare le seguenti caratteristiche:

- **Appropriatezza**: le funzioni fornite dal software sono appropriate per i compiti ed obiettivi prefissati;
- **Accuratezza**: il software deve fornire risposte concordanti o i precisi effetti richiesti;
- **Interoperabilità**: capacità di interagire ed operare con uno o più sistemi specificati;
- **Conformità**: il prodotto deve essere in grado di aderire a standard, convenzioni e regolamentazioni;
- **Sicurezza**: è la capacità del software di proteggere informazioni e dati, negandone l'accesso a persone o sistemi non autorizzati e invece fornirlo a chi ne è effettivamente autorizzato.

A.1.1.2 Affidabilità

È la capacità del prodotto software di mantenere uno specificato livello di prestazioni quando usato in date condizioni.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **Maturità**: capacità di un prodotto software di evitare che si verifichino errori, malfunzionamenti o siano prodotti risultati non corretti;

- **Tolleranza agli errori**: il software deve mantenere dei predeterminati livelli di prestazioni anche in caso di un uso scorretto o di malfunzionamenti;
- **Recuperabilità**: capacità di ripristinare il livello appropriato di prestazioni a seguito di un malfunzionamento o di un errore. La caratteristica di recuperabilità è valutata dall'arco di tempo nel quale il software può risultare non accessibile;
- **Aderenza**: è la capacità di aderire a standard, regole e convenzioni riguardanti l'affidabilità.

A.1.1.3 Efficienza

Capacità del prodotto software di eseguire le proprie funzioni minimizzando il tempo necessario e sfruttando al meglio le risorse di cui necessita.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **Comportamento rispetto al tempo**: è la capacità di fornire adeguati tempi di risposta, elaborazione e velocità di attraversamento, sotto condizioni determinate;
- **Utilizzo delle risorse**: è la capacità di utilizzo di quantità e tipo di risorse in maniera adeguata;
- **Conformità**: è la capacità di aderire a standard e specifiche sull'efficienza$_G$.

A.1.1.4 Usabilità

È la capacità del prodotto software di essere capito, appreso, usato e ben accettato dall'utente, anche quando usato sotto condizioni specificate.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **Comprensibilità**: esprime la facilità di comprensione dei concetti, funzionalità e utilizzo da parte dell'utente;
- **Apprendibilità**: è la capacità di ridurre l'impegno richiesto agli utenti per imparare ad usare l'applicazione;
- **Operabilità**: capacità di mettere in condizione gli utenti di farne uso per i propri scopi e controllarne l'uso;
- **Attrattivà:** è la capacità del software di essere piacevole all'uso per l'utente;
- **Conformità**: è la capacità del software di aderire a standard o convenzioni relativi all'usabilità.

A.1.1.5 Manutenibilità

La manutenibilità è la capacità del software di essere modificato, includendo correzioni, miglioramenti o adattamenti. Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **Analizzabilità**: rappresenta la facilità con la quale è possibile analizzare il codice per localizzare un errore nello stesso;
- **Modificabilità**: è la capacità del prodotto software di permettere l'implementazione di una modifica;
- **Stabilità**: capacità del software di evitare effetti inaspettati a seguito di modifiche errate;
- **Testabilità**: è la capacità di essere facilmente testato per validare le modifiche apportate al software.

A.1.1.6 Portabilità

La portabilità è la capacità del software di essere trasportato da un ambiente$_G$ di lavoro ad un altro. Ambiente che può variare dall'hardware al software. Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **Adattabilità**: la capacità del software di essere adattato per differenti ambienti senza dover applicare modifiche;
- **Installabilità**: capacità del software di essere installato in un diverso ambiente;
- **Conformità**: la capacità del prodotto software di aderire a standard e convenzioni relative alla portabilità;
- **Sostituibilità**: capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti nello stesso ambiente.

A.1.2 Metriche per la qualità esterna

Servono a misurare i comportamenti del software sulla base dei test, in funzione degli obiettivi stabiliti. Viene rilevata tramite analisi dinamica$_G$.

A.1.3 Metriche per la qualità interna

Metriche che si applicano al software non eseguibile durante le fasi di progettazione e codifica. Permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che venga realizzato un eseguibile. Grazie alle misure effettuate tramite le metriche interne è possibile prevedere il livello di qualità esterna e di qualità in uso del prodotto finale, poiché entrambe vengono influenzate dalla qualità interna.
Viene rilevata tramite analisi statica$_G$.

A.1.4 Metriche per la qualità in uso

La qualità in uso rappresenta il punto di vista dell'utente sul software. Il livello di qualità viene raggiunto dopo che sono stati raggiunti i livelli nella qualità esterna e interna. La qualità in uso, quindi, permette di abilitare specificati utenti ad ottenere specificati obiettivi con efficacia$_G$, produttività, sicurezza e soddisfazione.

A.2 ISO/IEC 15504

Il modello ISO/IEC 15504, meglio conosciuto come SPICE (*Software Process Improvement and Capability Determination*), è lo standard di riferimento per valutare in modo oggettivo la qualità dei processi nello sviluppo del software.
SPICE mette a disposizione una metrica per valutare diversi attributi per ogni processo ed assegna un valore quantificabile ad ognuno di questi in modo tale da rendere esplicito come poter migliorare tale processo.

A.2.1 Classificazione dei processi

SPICE definisce una serie di sei livelli utilizzati per classificare la capacità$_G$ e la maturità$_G$ del processo software. Ogni livello è caratterizzato dal soddisfacimento degli attributi associati:

1. **Incompleto**: il processo non è stato implementato o non ha raggiunto il successo desiderato;
2. **Eseguito**: il processo è implementato e ha realizzato il suo obiettivo (conformità);
   Attributi:
   - Process Performance.
3. **Gestito**: il processo è gestito e il prodotto finale è verificato, controllato e manutenuto (affidabilità);
   Attributi:
   - Performance Management;
   - Work Product Management.
4. **Stabilito**: il processo è basato sullo standard di processo (standardizzazione);
   Attributi:
   - Process Definition;
   - Process Deployment.
5. **Predicibile**: il processo è consistente e rispetta limiti definiti (strategico);
   Attributi:
   - Process Measurement;
   - Process Control.
6. **Ottimizzato**: il processo segue un miglioramento continuo per rispettare tutti gli obiettivi di progetto;
   Attributi:
   - Process Innovation;
   - Process Optimization.

Ogni attributo riceve una valutazione nella seguente scala, andando a definire il rispettivo livello di capacità del processo:

- N: non raggiunto (0 - 15%);
- P: parzialmente raggiunto (>15% - 50%);
- L: largamente raggiunto (>50%- 85%);
- F: pienamente raggiunto (>85% - 100%).

A.2.2 Linee guida

Lo standard definisce delle linee guida per effettuare delle stime, tali linee guida sono:

- processi di misurazione, descritti nel *Piano di Progetto* ;
- modello di misurazione, descritto in questo documento;
- strumenti di misurazione, descritti nelle *Norme di Progetto*.

A.2.3 Competenze

Per effettuare le misurazioni necessarie per determinare la qualità raggiunta, sono necessarie delle competenze. La mancanza di esperienza degli elementi del *TeamAFK* fa sì che nessun membro possieda queste abilità, rendendo così impossibile la piena adesione allo standard. Tuttavia, ogni componente è chiamato a studiare SPICE e ad applicare al meglio le indicazioni descritte in questo documento e nelle *Norme di Progetto*, al fine di perseguire un livello di qualità accettabile.