
# Norme di Progetto

## Jawa Druids

| | |
|---|---|
| **Versione** | 2.0.0 |
| **Data approvazione** | 2021-03-15 |
| **Responsabile** | Mattia Cocco |
| **Redattori** | Andrea Cecchin<br>Emma Roveroni |
| **Verificatori** | Margherita Mitillo<br>Andrea Dorigo<br>Igli Mezini |
| **Stato** | Approvato |
| **Lista distribuzione** | Jawa Druids<br>Prof. Tullio Vardanega<br>Prof. Riccardo Cardin |
| **Uso** | Interno |

## Sommario

Il documento redatto riferisce le regole, gli strumenti e le convenzioni a cui il gruppo *Jawa Druids* ha stabilito di attenersi e seguire per l'intera durata dello sviluppo del progetto.

# Registro delle modifiche

| Versione | Data | Autore | Ruolo | Modifica | Verificatore |
|---|---|---|---|---|---|
| v2.0.7 | 2021-04-21 | Alfredo Graziano | *Amministratore* | *Aggiunte § 2.2.6.6, § 2.2.6.7, § 2.2.6.8, § 2.2.6.9 e § 2.2.6.10* | Igli Mezini |
| v2.0.6 | 2021-04-21 | Alfredo Graziano | *Amministratore* | *Aggiunte § 2.2.7.6, § 2.2.7.7 e § 2.2.7.8* | Igli Mezini |
| v2.0.5 | 2021-04-20 | Igli Mezini | *Amministratore* | *Aggiunte § 2.2.5.6.4 e § 2.2.5.6.5* | Alfredo Graziano |
| v2.0.4 | 2021-04-13 | Emma Roveroni | *Amministratore* | *Modificata § 3.4.5.1* | Margherita Mitillo |
| v2.0.3 | 2021-04-10 | Emma Roveroni | *Amministratore* | *Modificata § 3.1.7.5 e § 3.1.8.3* | Margherita Mitillo |
| v2.0.2 | 2021-04-07 | Emma Roveroni | *Amministratore* | *Modificata § 2.2.6.4* | Margherita Mitillo |
| v2.0.1 | 2021-03-20 | Emma Roveroni | *Amministratore* | *Aggiunta § 4.3* | Margherita Mitillo |
| v2.0.0 | 2021-03-15 | Mattia Cocco | *Responsabile* | *Approvazione del documento per RP* | - |
| v1.2.0 | 2021-03-05 | - | - | *Revisione complessiva del documento* | Andrea Dorigo |
| v1.1.2 | 2021-03-03 | Emma Roveroni | *Amministratore* | *Aggiunte § 2.2.7.7 e § 3.4.5* | Margherita Mitillo |
| v1.1.1 | 2021-03-01 | Emma Roveroni | *Amministratore* | *Aggiunta § 2.2.6.4 e aggiornata § 2.2.6.5* | Margherita Mitillo |
| v1.1.0 | 2021-02-28 | - | - | *Revisione complessiva del documento* | Andrea Dorigo |

| | | | | | |
|---|---|---|---|---|---|
| v1.0.2 | 2021-02-27 | Andrea Cecchin | *Amministratore* | *Stesura § 2.2.5.4 e § 2.2.5.6* | Igli Mezini |
| v1.0.1 | 2021-02-20 | Emma Roveroni | *Amministratore* | *Aggiornate § 3.1.7.2 e § 3.2.2.1* | Margherita Mitillo |
| v1.0.0 | 2021-12-29 | Igli Mezini | *Responsabile di progetto -* | *Approvazione del documento per RR* | - |
| v0.9.0 | 2021-01-09 | - | - | *Revisione Finale del documento* | Margherita Mitillo |
| v0.8.0 | 2021-01-08 | - | - | *Verificato capitolo 7* | Margherita Mitillo |
| v0.7.0 | 2021-01-08 | - | - | *Verificate § 2.2.5, § 2.2.6, § 2.2.7, § 6* | Andrea Dorigo |
| v0.6.0 | 2021-01-07 | - | - | *Verificate § 3.2, § 4.2, § 4.4* | Margherita Mitillo |
| v0.5.0 | 2021-01-07 | - | - | *Verificate § 3.4, § 3.5, § 5* | Emma Roveroni |
| v0.4.7 | 2021-01-06 | Mattia Cocco | *Amministratore* | *Aggiunto capitolo 7* | - |
| v0.4.6 | 2021-01-05 | Alfredo Graziano | *Amministratore* | *Aggiunte § 3.4 e § 3.5* | - |
| v0.4.5 | 2021-01-05 | Mattia Cocco | *Amministratore* | *Aggiunto capitolo 4.4* | - |
| v0.4.4 | 2021-01-03 | Andrea Cecchin | *Amministratore* | *Aggiunte § 2.2.5, § 2.2.6, § 2.2.7* | - |
| v0.4.3 | 2021-01-02 | Andrea Dorigo | *Amministratore* | *Aggiunta § 3.2* | - |
| v0.4.2 | 2021-01-02 | Mattia Cocco | *Amministratore* | *Aggiunte § 2.2.4.8, § 3.4.1* | - |
| v0.4.1 | 2021-01-01 | Alfredo Graziano | *Amministratore* | *Aggiunto capitolo 6* | - |
| v0.4.0 | 2020-12-31 | - | - | *Verificate § 3.3 e § 4.1* | Emma Roveroni |

| | | | | | |
|---|---|---|---|---|---|
| v0.3.4 | 2020-12-30 | Alfredo Graziano | *Amministratore* | *Aggiunta § 4.2.5* | - |
| v0.3.3 | 2020-12-30 | Mattia Cocco | *Amministratore* | *Aggiunte § 3.3, § 5* | - |
| v0.3.2 | 2020-12-30 | Igli Mezini | *Amministratore* | *Aggiunte § 4.2.1, § 4.2.2, § 4.2.3, § 4.2.4, § 4.2.6* | - |
| v0.3.1 | 2020-12-29 | Igli Mezini | *Amministratore* | *Aggiunte § 4.1.1, § 4.1.2, § 4.1.3, § 4.1.4* | - |
| v0.3.0 | 2020-12-15 | - | - | *Verificata § 3.1* | Andrea Dorigo |
| v0.2.0 | 2020-12-11 | - | - | *Verificate § 2.1 e § 2.2* | Margherita Mitillo |
| v0.1.0 | 2020-12-10 | - | - | *Verificato capitolo 1* | Margherita Mitillo |
| v0.0.9 | 2020-12-08 | Andrea Dorigo | *Amministratore* | *Aggiunte § 3.1.10.2, § 3.1.6* | - |
| v0.0.8 | 2020-12-03 | Igli Mezini | *Amministratore* | *Aggiunte § 3.1.10, § 3.1.11* | - |
| v0.0.7 | 2020-12-01 | Igli Mezini | *Amministratore* | *Aggiunte § 3.1.7, § 3.1.8, § 3.1.9* | - |
| v0.0.6 | 2020-11-30 | Andrea Cecchin | *Amministratore* | - | *Aggiunta § 2.2* - |
| v0.0.5 | 2020-11-29 | Igli Mezini | *Amministratore* | *Aggiunte § 3.1.4, § 3.1.5, § 3.1.6* | - |
| v0.0.4 | 2020-11-28 | Igli Mezini | *Amministratore* | *Aggiunte § 3.1.1, § 3.1.2, § 3.1.3* | - |
| v0.0.3 | 2020-11-26 | Andrea Cecchin | *Amministratore* | *Aggiunte § 2.1.2, § 2.1.3, § 2.1.4* | - |
| v0.0.2 | 2020-11-25 | Andrea Cecchin | *Amministratore* | *Aggiunte § 1 e § 2.1.1* | - |
| v0.0.1 | 2020-11-24 | Andrea Cecchin | *Amministratore* | *Prima stesura del documento* | - |

# Indice

# Elenco delle figure

# 1 Introduzione

## 1.1 Scopo del documento

Lo scopo del documento è quello di formalizzare tutte le regole e procedure$_G$ fondamentali che ciascun membro di *Jawa Druids* si impegna a rispettare per tutta la durata dello sviluppo del progetto. Le norme verranno aggiunte passo dopo passo a seguito di un'attenta analisi e concordate all'interno del gruppo preventivamente. L'attuazione di queste regole e norme permette di ottenere un'organizzazione uniforme ed efficiente dei file prodotti.

## 1.2 Scopo del prodotto

In seguito alla pandemia del virus COVID-19 è nata l'esigenza di limitare il più possibile i contatti fra le persone, specialmente evitando la formazione di assembramenti. Il progetto *GDP: Gathering Detection Platform* di *Sync Lab* ha pertanto l'obiettivo di **creare una piattaforma in grado di rappresentare graficamente le zone potenzialmente a rischio di assembramento, al fine di prevenirlo.** Il prodotto finale è rivolto specificatamente agli organi amministrativi delle singole città, cosicché possano gestire al meglio i punti sensibili di affollamento, come piazze o siti turistici. Lo scopo che il software intende raggiungere non è solo quello della rappresentazione grafica real-time ma anche quella di poter riuscire a prevedere assembramenti in intervalli futuri di tempo.

Al tal fine il gruppo *Jawa Druids* si prefigge di sviluppare un prototipo software in grado di acquisire, monitorare ed analizzare i molteplici dati provenienti dai diversi sistemi e dispositivi, a scopo di identificare i possibili eventi che concorrono all'insorgere di variazioni di flussi di utenti. Il gruppo prevede inoltre lo sviluppo di un'applicazione web da interporre fra i dati elaborati e l'utente, per favorirne la consultazione.

## 1.3 Glossario

All'interno della documentazione viene fornito un *Glossario*, con l'obiettivo di assistere il lettore specificando il significato e contesto d'utilizzo di alcuni termini strettamente tecnici o ambigui, segnalati con una $G$ a pedice.

## 1.4 Riferimenti

### 1.4.1 Riferimenti normativi

- *Norme di Progetto*;
- *Piano di Progetto 2.0.0.*

### 1.4.2 Riferimenti informativi

- *IEEE Recommended Practice for Software Requirements Specifications:* https://ieeexplore.ieee.org/document/720574
- *Standard ISO/IEC 12207:1995:* https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf
- *Seminario per approfondimenti tecnici del capitolato$_G$ C3:* https://www.math.unipd.it/~tullio/IS-1/2020/Progetto/ST1.pdf
- *Definizioni dei diagrammi delle classi:* https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf
- *Definizioni dei diagrammi di attività:* https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20di%20Attivit%c3%a0_4x4.pdf
- *Definizioni dei diagrammi dei package:* https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20dei%20Package_4x4.pdf

# 2 Processi Primari

## 2.1 Fornitura

### 2.1.1 Scopo

La fornitura secondo lo standard ISO/IEC 12207:1995 descrive le attività$_G$ e i compiti svolti dal fornitore al fine di sviluppare un prodotto soddisfacente e che rispetti appieno le richieste del committente$_G$. Durante questa fase si prevede la compilazione di diversi documenti, i quali verranno inviati al committente$_G$ per guadagnare la possibilità di lavorare al progetto offerto dall'azienda *Sync Lab*. Il fornitore esegue un'attività$_G$ di analisi e stesura dello *Studio di Fattibilità*, documento che rileva i rischi e le criticità riscontrate nella richiesta di appalto. Si definisce inoltre un accordo contrattuale con il proponente$_G$ mediante il quale si regolano i rapporti con l'azienda, la consegna e la manutenzione del prodotto sviluppato.

### 2.1.2 Studio di Fattibilità

Lo *Studio di Fattibilità* consiste nell'analisi e nella valutazione sistematica delle caratteristiche, dei costi, e dei possibili risultati di un progetto sulla base di una preliminare idea di massima. A seguito della presentazione dei capitolati d'appalto da parte di ogni proponente$_G$ avvenuta il 2020-11-05, il *Responsabile di Progetto* si è impegnato a programmare incontri con tutti i componenti del gruppo *Jawa Druids* per valutare le scelte di ogni membro e attuare così un primo scambio di idee. Una volta individuato il capitolato d'interesse gli *Analisti* hanno provveduto alla stesura dello *Studio di Fattibilità*, i quali hanno fornito un'analisi accurata dei capitolati presentati. Nella stesura dello *Studio di Fattibilità* per ogni capitolato$_G$ si riporterà:

- informazioni generali: informazioni riguardanti il proponente$_G$;
- descrizione del capitolato$_G$: sintesi del progetto da sviluppare;
- finalità del progetto: finalità richieste dal capitolato$_G$ d'appalto;
- tecnologie interessate: tecnologie che verranno utilizzate nello svolgimento del capitolato$_G$;
- aspetti positivi: aspetti favorevoli alla scelta del capitolato$_G$;
- criticità e fattori di rischio: problematiche che potrebbero sorgere durante lo svolgimento del capitolato;
- conclusioni: accettazione o rifiuto del capitolato$_G$ in base alle informazioni illustrate precedentemente e anche all'interesse dimostrato da ogni membro nel gruppo.

### 2.1.3 Altra documentazione da fornire

Oltre allo *Studio di Fattibilità* vengono consegnati altri documenti all'azienda *Sync Lab* ed ai committenti *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*. Questi documenti sono necessari al fine di tracciare le attività$_G$ di Analisi, Pianificazione, Verifica, Validazione e Controllo di Qualità per assicurare una completa trasparenza durante tutta la durata del ciclo di vita del progetto. I documenti in questione sono:

- *Analisi dei Requisiti 3.0.0*: identifica e dettaglia in modo completo ed esaustivo i requisiti$_G$ del sistema descritto nel capitolato che il fornitore si impegna a soddisfare;
- *Piano di Qualifica 2.0.0*: illustra la strategia complessiva di verifica e validazione proposta dal fornitore per pervenire al collaudo del sistema con la massima efficienza ed efficacia;
- *Piano di Progetto 2.0.0*: presenta l'organigramma dettagliato del fornitore, lo schema proposto per l'assegnazione e la rotazione dei ruoli di progetto, l'impegno complessivo previsto per ogni ruolo e per ogni individuo, l'analisi dei rischi, la pianificazione di massima per la realizzazione del prodotto, e il corrispondente conto economico preventivo;
- *Proof of Concept$_G$ e Technology Baseline$_G$*: il Proof of Concept$_G$ dimostra una baseline$_G$ per lo sviluppo del prodotto, mentre la Technology Baseline$_G$ definisce le tecnologie utilizzate.

Alla documentazione appena illustrata il gruppo *Jawa Druids* allegherà inoltre una lettera di presentazione con la quale si formalizza l'impegno nel portare al termine il capitolato prescelto entro i termini definiti nella lettera e rispettandone i requisiti$_G$ minimi.

### 2.1.4 Strumenti

Di seguito sono riportati gli strumenti impiegati dal gruppo durante il progetto per il processo di fornitura.

#### 2.1.4.1 Documenti Google

Questo strumento viene utilizzato per la realizzazione di documenti in cui più persone possono lavorarci contemporaneamente osservando le modifiche in tempo reale.

https://docs.google.com

#### 2.1.4.2 Fogli Google

Questo strumento viene utilizzato per la realizzazione di grafici e fogli di calcolo in cui più persone possono lavorarci contemporaneamente osservando le modifiche in tempo reale.

https://www.google.com/sheets

## 2.2 Sviluppo

### 2.2.1 Scopo

Il processo di sviluppo contiene tutte le attività$_G$ che riguardano la produzione del software richiesto dal cliente, in particolare *Analisi dei Requisiti*$_G$, design, codifica, integrazione, test e installazione.

### 2.2.2 Descrizione

Di seguito vengono elencate le varie attività$_G$ che caratterizzano tale processo:

- Analisi dei requisiti$_G$;
- Progettazione architetturale;
- Codifica del software.

### 2.2.3 Prospettive

Le prospettive alla fine della stesura del processo in questione sono le seguenti:

- individuare e stabilire gli obbiettivi di sviluppo;
- individuare e stabilire i vincoli tecnologici;
- individuare e stabilire i vincoli di design;
- produrre un prodotto finale che rispecchi gli obiettivi imposti nello sviluppo e che superi i test e i controlli di qualità stabiliti dal proponente$_G$.

### 2.2.4 Analisi dei requisiti

#### 2.2.4.1 Scopo

L'*Analisi dei Requisiti*$_G$ viene redatto dagli *Analisti*, lo scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti$_G$ che devono essere soddisfatti dal software sviluppato.
Gli obiettivi della stesura dell'*Analisi dei Requisiti* sono:

- stabilire lo scopo nello sviluppo del prodotto;
- definire riferimenti precisi ed affidabili ai *Progettisti*;

- stabilire i requisiti$_G$ e le funzionalità concordate con il cliente;
- individuare per i *Verificatori* riferimenti per le attività$_G$ di controllo dei test.

#### 2.2.4.2 Descrizione

I requisiti$_G$ possono essere individuati in diverse fonti, quali:

- *Capitolato$_G$ d'Appalto*: i requisiti$_G$ sono stati individuati attraverso la lettura del documento fornito dal proponente$_G$ *Sync Lab* sul capitolato proposto;
- *Verbali Interni*: attraverso le riunioni attuate internamente dagli *Analisti* sono emersi vari requisiti$_G$;
- *Verbali Esterni*: attraverso contatti e discussioni effettuate con il responsabile aziendale Fabio Pallaro sono emersi requisiti$_G$, i quali vi sarà assegnato un codice presente nella tabella dei tracciamenti.

#### 2.2.4.3 Prospettive

L'obiettivo dell'*Analisi dei Requisti* è quello di redigere un documento che racchiuda al suo interno tutti i requisiti$_G$ richiesti dal proponente$_G$.

#### 2.2.4.4 Struttura

L'*Analisi dei requisiti$_G$* è strutturato nel seguente modo:

- **Introduzione**: in questa sezione si introduce lo scopo del documento e riferimenti a documenti esterni;
- **Descrizione generale**: il prodotto viene descritto, si individuano le fasi generali del progetto e l'utenza a cui è destinato il prodotto;
- **Casi d'uso$_G$**: si elencano tutti i casi d'uso$_G$ individuati, questo è utile al tracciamento dei requisiti e permette di individuare tutte le funzionalità del progetto;
- **Requisiti$_G$**: utilizzando i casi d'uso descritti e tutti i documenti sopra indicati si dettagliano tutti i requisiti$_G$ obbligatori, facoltativi e desiderabili da implementare.

#### 2.2.4.5 Classificazione dei requisiti

I requisiti$_G$ sono stati individuati utilizzando la seguente codifica:

**RS**[**classificazione**][**tipo_di_requisito**][**codice_requisito**]

La descrizione della classificazione è la seguente:

- **RS**: è l'acronimo per Requisito$_G$ Specifico;
- **Classificazione**: individua la classificazione del requisito$_G$:
  - Funzionale: indicato dalla lettera "F";
  - Qualità: indicato dalla lettera "Q";
  - Vincolo: indicato dalla lettera "V";
  - Prestazionale: indicato dalla lettera "P".
- **Tipo_di_requisito**$_G$: individua la tipologia di requisito$_G$:
  - Obbligatorio: indicato con la lettera "O" individua un requisito$_G$ essenziale allo sviluppo del progetto e necessario al suo completamento;
  - Desiderabile: indicato con la lettera "D" individua un requisito$_G$ utile al prodotto e che dà valore aggiunto ad esso, ma non essenziale al suo completamento;
  - Facoltativo: indicato con la lettera "F" individua un requisito$_G$ che può essere sviluppato, ma può anche non essere completato.
- **Codice_requisito**: è rappresentato da un codice identificativo univoco nella forma gerarchica padre/figlio.

Ogni requisito$_G$ è strutturato nella tabella nel seguente modo:

- **Identificativo**: individua univocamente il requisito$_G$;
- **Descrizione**: descrizione del requisito$_G$;
- **Tipo di requisito**$_G$: individua la tipologia di requisito$_G$ obbligatorio, desiderabile e facoltativo;
- **Fonte**: ogni requisito$_G$ è ricavato da una delle seguenti fonti:
  - Capitolato$_G$: individua il documento del capitolato$_G$;
  - Interno: il requisito$_G$ è stato individuato dagli analisti;
  - Verbale: si tratta del documento in riferimento alla discussione con il proponente$_G$;
  - Casi d'uso$_G$: il requisito$_G$ è stato individuato dal caso d'uso$_G$ individuato con il proprio codice.

#### 2.2.4.6 Classificazione dei casi d'uso

I casi d'uso$_G$ individuano le iterazioni tra il sistema ed un attore. I casi d'uso$_G$ vengono identificati nel seguente modo:

**UC[codice_Padre].[codice_Figlio]**

La descrizione della classificazione è la seguente:

- **UC**: è l'acronimo per User Case$_G$, la parola inglese che si traduce in Casi D'uso$_G$;
- **Codice_Padre.Codice_Figlio**: individua un codice univoco per ogni caso d'uso$_G$ nella forma gerarchica padre/figlio;

Ogni caso d'uso$_G$ si struttura nel seguente modo:

- **Principali**:
  - **Attori**: definisce una persona o un elemento esterno che interagisce con il sistema per avviare il caso d'uso$_G$;
  - **Descrizione**: breve descrizione del caso d'uso$_G$;
  - **Scenario principale**: descrive le azioni necessarie al completamento del caso d'uso$_G$;
  - **Precondizione**: individua le condizioni necessarie per l'avvio del caso d'uso$_G$;
  - **Postcondizione**: individua le condizioni del sistema al completamento del caso d'uso$_G$.
- **Aggiuntivi**:
  - **Input**: individua file da inserire nel sistema da parte dell'attore;
  - **Output**: Individua file in uscita dal sistema per l'attore;
  - **Estensioni**: individua le condizioni nel quale viene utilizzato un caso d'uso$_G$ esterno, tale che aumenti le funzionalità del caso d'uso$_G$ sotto osservazione;
  - **Generalizzazioni**: individua la generalizzazione del caso d'uso$_G$ in sotto casi figli;
  - **Inclusioni**: individuano altri casi d'uso$_G$ che vengono utilizzati per compiere il caso d'uso$_G$ in osservazione.

#### 2.2.4.7 Qualità dei requisiti

Ogni requisito$_G$ deve rispettare le seguenti qualità:

- devono essere correttamente descritti;

- non devono essere ambigui, ogni requisito$_G$ fornirà un'unica interpretazione;
- devono essere completi, cioè descrivere in modo completo e in tutte le sue parti la funzionalità da implementare;
- ogni requisito$_G$ deve essere consistente, cioè non deve avere conflitti con altri requisiti$_G$ individuati;
- devono essere modificabili, cioè ogni requisito$_G$ nel corso dello sviluppo del progetto può essere rivalutato e modificato e bisogna mantenere uno storico dei cambiamenti;
- ogni requisito$_G$ deve essere tracciabile, cioè ogni requisito$_G$ deve essere tracciabile ad ogni suo test o codice di implementazione o alla sua origine;
- ogni requisito$_G$ deve essere classificato per importanza o stabilità;
- ogni requisito$_G$ deve essere verificabile, cioè deve essere possibile una sua verifica attraverso un processo nel quale una persona o una macchina può controllarlo.

#### 2.2.4.8 Metriche

- Con **MQPD01** si intende la **Totalità delle implementazioni**. Indice riportante l'interezza del prodotto software, rispetto ai requisiti$_G$ posti, mediante un valore percentuale.:

$$\mathbf{T}=(1-\frac{RnI}{RI})*\mathbf{100}$$

  Dove:

  - **T** sta per *Totalità*, riferito ai requisiti$_G$ da implementare;
  - **RnI** sta per *Requisito$_G$ non Implementato*;
  - **RI** sta per *Requisito$_G$ Implementato*.

  I range accettabili per il risultato di **T** sono così suddivisi:

  - $90\% < \mathbf{T} \leq 100\%$ indica che la copertura dei requisiti$_G$ proposti è quasi totale;
  - $80\% < \mathbf{T} \leq 90\%$ indica che la copertura dei requisiti$_G$ proposti è sufficiente, buona;
  - $\mathbf{T} \leq 80\%$ indica che la copertura dei requisiti$_G$ proposti è insufficiente.

- Con **MQPS07** viene inteso un valore, in formato percentuale, riferito ai requisiti$_G$ *obbligatori* implementati: **PROI: Percentuale Requisiti Obbligatori Implementati**.

$$\textbf{PROI} = fracROIROT\textbf{*100}$$

Dove:

- **ROI:** requisiti obbligatori implementati;
- **ROT:** requisiti obbligatori totali.

I range dei valori che deve assumere il **PROI** sono così suddivisi:

- **valore preferibile:** 100%;
- **valore accettabile:** 100%.

## 2.2.5 Progettazione

### 2.2.5.1 Scopo

La Progettazione è un'attività$_G$ svolta dai *Progettisti*, nella quale si individuano, tramite l'*Analisi dei Requisiti 3.0.0*, le caratteristiche che il prodotto deve avere per soddisfare tutti i requisiti$_G$ richiesti dal proponente$_G$. Lo scopo è quello di determinare la soluzione migliore per assolvere ogni requisito$_G$ individuato.

### 2.2.5.2 Aspettative

Precedentemente alla realizzazione dell'architettura del sistema dovranno essere definite:

- le tecnologie da utilizzare;
- i vincoli strutturali richiesti dal proponente;
- il Proof of Concept$_G$, cioè una bozza eseguibile del prodotto che individui lo studio svolto in preparazione all'architettura da implementare.

Alla conclusione della stesura della progettazione si ha come risultato la realizzazione dell'architettura del software da sviluppare. Tale architettura è necessaria ai programmatori per individuare le istruzioni necessarie a sviluppare il prodotto finito.

### 2.2.5.3 Descrizione

La progettazione prevede inizialmente la realizzazione del Proof of Concept$_G$ della Technology Baseline$_G$, che, successivamente, viene approfondito e redatto nel documento tecnico allegato alla Product Baseline$_G$. La progettazione segue l'*Analisi dei Requisiti 3.0.0*: infatti, se l'Analisi è l'espansione del problema e l'identificazione delle parti che lo compongono per l'individuazione del dominio applicativo, allora la progettazione è il procedimento inverso, cioè si trova un modo per collegare tutte le parti assieme, specificando le funzionalità di tutti i sottosistemi e raggiungendo così un'unica soluzione.

#### 2.2.5.4 Qualità dell'architettura

L'architettura del progetto viene definita dai *Progettisti* per individuare una logica corretta allo sviluppo del prodotto. Ogni modulo definito all'interno dell'architettura deve essere identificabile in modo chiaro e riusabile. L'architettura dovrà rispettare delle caratteristiche per raggiungere una qualità adeguata:

- dovrà soddisfare i requisiti identificati nel documento *Analisi dei Requisiti 3.0.0* e poter essere modulabile nel caso essi vengano modificati o aggiunti;
- dovrà essere capita dagli stakeholders$_G$ e quindi dovrà essere fatta una stesura chiara e comprensibile con il tracciamento sui requisiti annesso;
- dovrà poter mantenere un grado di funzionamento adeguato anche in caso di situazioni erronee improvvise;
- dovrà essere possibile applicare modifiche a costi ridotti in caso i requisiti vengano modificati o evoluti;
- dovrà essere modellata in modo tale da poter riutilizzare alcune delle sue parti;
- dovrà comprendere tutti i requisiti e soddisfarli in modo da eliminare sprechi di tempo e spazio;
- dovrà svolgere tutti i suoi compiti al suo utilizzo;
- dovrà essere sicura e la sua manutenzione dovrà avvenire in tempo ridotti cosicché da garantire un servizio di funzionamento il più continuo possibile;
- dovrà essere semplice ed evitare la complessità introducendo solo il necessario.

#### 2.2.5.5 Attività di progettazione

Le attività da svolgere durante il progetto sono suddivise per obiettivi da raggiungere e rientrano nello specifico arco temporale definito nel *Piano di Progetto 2.0.0*. Consistono in progettazione architetturale, progettazione dettagliata e codifica, durante i quali verranno affrontate la Technology Baseline$_G$ e la Product Baseline$_G$,

- **Progettazione architetturale**

  Durante questo periodo vengono definiti i componenti del sistema, ovvero i task da eseguire per completare con successo il periodo.

  - identificazione delle componenti coinvolte nel sistema;
  - organizzazione dei ruoli, responsabilità e interazioni di ogni componente;

- definizione delle interazioni tra le componenti tra loro e con l'ambiente;
- creazione e documentazione dei test di integrazione all'interno del *Piano di Qualifica 2.0.0*.

- **Progettazione di dettaglio**

  Le parti sono suddivise per determinare cosa dovrà fare il codice nel particolare, in modo da assegnare compiti indivisibili ai Programmatori.

  - definizione degli strumenti di verifica per le unità e per i moduli;
  - definizione dei ruoli che coinvolgono rispettivamente le unità e i moduli;
  - definizione delle interazioni che coinvolgono le unità;
  - suddivisione delle componenti individuate in unità;
  - definizione dell'unità come insieme dei moduli.

#### 2.2.5.6 Diagrammi UML 2.0

L'utilizzo dei diagrammi UML 2.0 è necessario per rendere più chiare le scelte progettuali adottate. I diagrammi che verranno utilizzati sono:

- **diagrammi dei package**$_G$: descrivono graficamente le dipendenze tra diversi package all'interno del sistema, i package sono raggruppamenti di classi in unià;
- **diagrammi delle classi**: rappresentano graficamente un modello astratto delle classi in descrizione e le loro realzioni, ogni classe viene individuata dai suoi attributi, il suo tipo e i suoi metodi;
- **diagrammi delle attività**: illustra graficamente la sequenza di attività per raggiungere un obiettivo da uno stato iniziale, essi possono descrivere la logica di un algoritmo;
- **diagrammi dei casi d'uso**: descrivono le funzionalità che il sistema offre;
- **diagrammi di sequenza**: rappresentano l'interazione logica tra diverse classi/oggetti;

**2.2.5.6.1 Diagrammi dei package:** Ogni package$_G$ viene individuato tramite un rettangolo con un'etichetta per il nome. Il package$_G$ contiene al suo interno i diagramma delle classi relative al package$_G$ e possibili sotto-package$_G$. Le dipendenze tra i package$_G$ vengono rappresentate attraverso una freccia tratteggiata che collega due package$_G$. Si dovrà evitare che le dipendenze dei package$_G$ così collegati formino una dipendenza ciclica.


Figura 2.1: Rappresentazione grafica dei diagrammi dei package

**2.2.5.6.2 Diagrammi delle classi:** Le classi vengono individuate da un rettangolo suddiviso in tre sezioni orizzontali. Ciascuna sezione partendo dall'alto individuano:

1. **Nome della classe**: deve essere univoco ed esplicativo. Il nome sarà scritto in lingua inglese, se la classe raffigurata è una classe astratta il suo nome verrà preceduto da $<< abstract >>$, mentre se è un'interfaccia verrà preceduto da $<< interface >>$.

2. **Attributi**: ogni attributo individua una variabile della classe. Gli attributi vengono scritti uno dopo l'altro sotto forma di lista. Ogni attributo sarà scritto nel seguente modo:

   **nomeAttributo : tipo**

   La descrizione della scrittura degli attributi è la seguente:

   - **nomeAttributo**: individua il nome della variabile, deve essere scritta in lingua inglese e con la lettera iniziale minuscola. Nel caso sia una variabile costante tutto il nome deve essere scritto in maiuscolo;
   - **tipo**: il tipo può essere semplice o definito da una classe creata dall'utente.

   Ogni attributo viene preceduto obbligatoriamente da un operatore:

   - $+$: indica la visibilità pubblica della variabile;
   - $-$: indica la visibilità privata della variabile;
   - $\#$: indica la visibilità protetta della variabile;
   - $\sim$: indica la visibilità package della variabile.

3. **Metodi**: indicano i metodi della classe, vengono disposti in lista uno dopo l'altro. I metodi vengono scritti nel seguente modo:

   **nomeMetodo (lista-param) : tipoR**

La descrizione della scrittura dei metodi è la seguente:

- **nomeMetodo**: individua il nome del metodo, deve essere univoco ed in lingua inglese;
- **lista-param**: indica la lista dei parametri formali del metodo che può essere composta da 0 o più parametri, ogni parametro è separato da una virgola con il successivo e sono scritti nel formato *nome : tipo*;
- **tipoR**: individua il tipo dell'oggetto di ritorno, può essere semplice o definito dall'utente.

Ogni metodo sarà preceduto dagli operatori di visibilità descritti precedentemente. Se il metodo sotto osservazione è un metodo astratto il suo nome viene preceduto da << *abstract* >>, mentre se è statico da << *static* >>.

Il collegamento tra i diagrammi delle classi viene effettuato con delle frecce che illustrano le dipendenze. I tipi di frecca vengono descritti di seguito:

- freccia normale, da classe 1 a classe 2: indica che gli oggetti delle due classi consividono una relazione statica;


Figura 2.2: Relazione di associazione tra classe 1 e classe 2

- freccia tratteggiata, da classe 1 a classe 2: indica che la definizione di una delle due classi fa riferimento alla definizione dell'altra;


Figura 2.3: Relazione di dipendenza tra classe 1 e classe 2

- freccia "a diamante" vuoto, da classe 1 a classe 2: indica una relazione del tipo "è parte di", cioè classe 2 è parte di classe 1, questo permette di aggregare più classi e creare una classe più complessa;


Figura 2.4: Relazione di aggregazione tra classe 1 e classe 2

- freccia "a diamante" piena, da classe 1 a classe 2: indica la composizione, è simile all'aggregazione, ma ha una relazione più forte sulla classe sub-ordinata perchè classe 2 non può esistere la classe 1;


Figura 2.5: Relazione di composizione tra classe 1 e classe 2

- freccia con punta vuota, da classe 1 a classe 2: indica la realizzazione/implementazione, cioè l'oggetto classe 1 implementa il comportamento, quindi nel nostro caso i metodi, che l'oggetto classe 2 specifica.

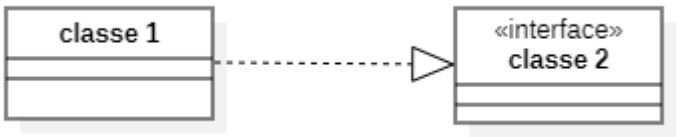


Figura 2.6: Relazione di interfaccie tra classe 1 e classe 2

Ad ogni freccia di collegamento viene indicato la molteplicità che ha e vengono indicate nel seguente modo:

- **1**: classe 1 possiede un'istanza di classe 2;
- **0...1**: classe 1 può possedere al massimo una istanza di classe 2;
- **0...*** classe 1 può possedere 0 o più istanze della classe 2;
- *: classe 1 può possedere più istanze della classe 2;
- **n**: classe 1 possiede $n$ istanze della classe 2.

**2.2.5.6.3 Diagrammi delle attività:** Il diagramma di attività è un tipo di diagramma che permette di descrivere un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite. I diagrammi vengono illustrati usando il seguente formalismo:

- **Nodo iniziale**: viene rappresentato da un pallino nero pieno ed indica l'inizio della attività dove viene generato un token;

Figura 2.7: Rappresentazione di un nodo iniziale

- **Attività**: viene rappresentata da un rettangolo, la sua descrizione deve essere breve e significativa per indicare l'azione svolta in quel punto del flusso dell'attività. Consuma e produce un token;

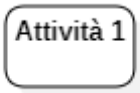


Figura 2.8: Rappresentazione di una attività

- **Nodo finale**: viene rappresentato da due cerchi concentrici di cui l'esterno è vuoto e quello interno è pieno. Indica il punto in cui termina l'esecuzione. Consuma un token;

Figura 2.9: Rappresentazione di un nodo finale

- **Nodo di fine flusso**: viene rappresentato attraverso un cerchio vuoto con una $X$ al centro. Indica la terminazione di un ramo dell'attività. Consuma un token;

Figura 2.10: Rappresentazione di un nodo di fine flusso

- **Sotto-attività**: viene rappresentata da un rettangolo con un tridente nell'angolo in basso a destra. Indica che l'attività viene rappresentata in un diagramma separato ed ogni sotto-attività ha un input ed un output;

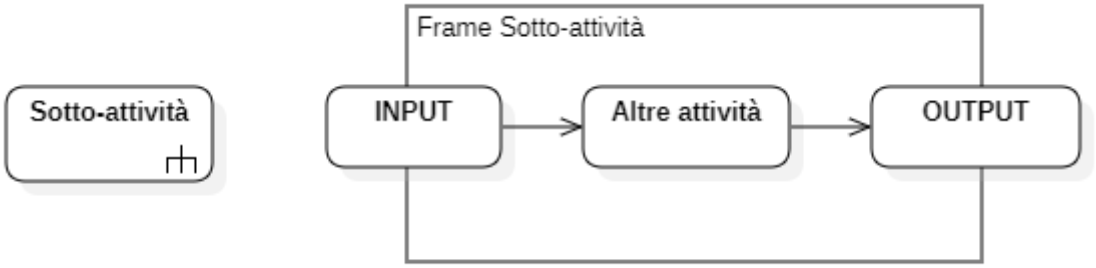


Figura 2.11: Rappresentazione di una sotto-attività

- **Branch**: viene rappresentato attraverso un rombo con una freccia in entrata e $n$ frecce in uscita. Ogni ramo in uscita deve avere una guardia, scritta nel formato [*guardia*]. Il branch definisce una scelta, tra i rami disponibili in usciti se ne può scegliere uno solo in base alla guardia. Consuma e produce un token;
- **Merge**: viene rappresentato da un rombo con $n$ frecce in entrata e una freccia in uscita. Individua il punto in cui i rami creati dal Branch tornano ad unirsi. Consuma e produce un token;

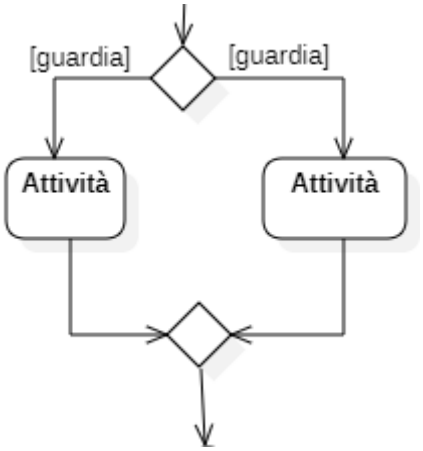


Figura 2.12: Rappresentazione di un branch e merge

- **Fork**: viene rappresentato da una linea nera marcata orizzontale o verticale. Indica il punto in cui avviene una paralizzazione delle attività da effettuare senza un limite

temporale. Il fork presenta una freccia in entrata e $n$ frecce in uscita. Consuma e produce un token;

- **Join**: viene rappresentato da una linea nera marcata orizzontale o verticale. Indica il punto in cui tutte le attività svolte in parallelo si sincronizzano. Il join preseta $n$ frecce in entrata e una freccia in uscita. Consuma e produce un token;

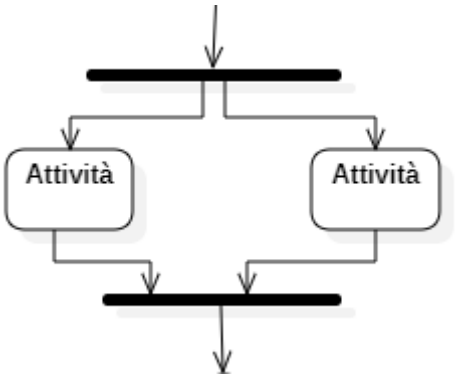


Figura 2.13: Rappresentazione di un fork e un join

- **Pin**: viene rappresentato da un quadrato con una freccia al suo interno nella direzione di input o output. Indica il parametro che viene inviato da un'attività all'altra, a fianco del quadrato viene scritto il tipo di parametro inviato;

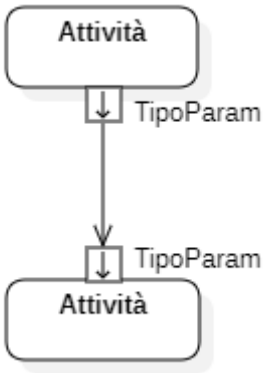


Figura 2.14: Rappresentazione di pin in un'attività

- **Segnali**: rappresentate da due figure "a incastro", la prima utilizzata per l'emissione del segnale, la seconda per la ricezione dello stesso. Il testo di descrizione all'interno delle figure deve essere breve e conciso e deve avere un prefisso $-signalsending-$ o $-signalreceipt-$ a secondo della figura in descrizione;

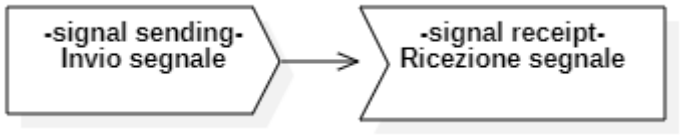


Figura 2.15: Rappresentazione di un segnale

- **Timeout**: viene rappresentato da una clessidra stilizzata. Indica due tipi di eventi uno è il timeout rappresentato dalla clessidra con una freccia entrante e uscente, serve a rappresentare un'attesa di tempo. L'altro tipo di evento è l'evento ripetuto, rappresentato da una clessidra con una freccia in uscita, serve ad indicare un'azione ripetuta.

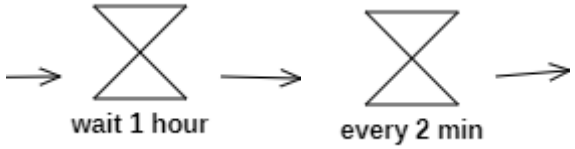


Figura 2.16: Rappresentazione di un evento timeout e uno ripetuto

**2.2.5.6.4 Diagrammi dei casi d'uso:** I diagrammi di dei casi d'uso descrivono le funzionalità del sistema attraverso una visione esterna. Nello specifico, un caso d'uso è un'insieme di scenari e sequenze di azioni che hanno lo stesso obiettivo per l'utente. Un diagramma non deve rappresentare alcun dettaglio implementativo, permettendo una descrizione delle funzionalità coinvolte con una visione esterna al sistema, come se fosse percepita dall'utente. Gli elementi presenti all'interno di un caso d'uso sono i seguenti:

- **Attore:** viene disegnato come un omino stilizzato e sotto viene posto il nome;
- **Caso d'uso:** rappresentato con un ovale e al suo interno viene inserita la descrizione dello stesso. Ogni caso d'uso viene associato ad un attore e viceversa per mezzo di una linea semplice;

Gli elementi contenuti all'interno di un diagramma in base alle funzionalità che devono essere rappresentate, le seguenti relazioni:

- **Associazione:** è la comunicazione diretta tra attore ed use case. Rappresenta la partecipazione dell'attore al caso d'uso a cui è collegato. Un'associazione viene rappresentata mediante una linea che collega l'attore al caso d'uso;
- **Inclusione:** L'inclusione è un legame diretto stretto tra due use case. Dati due casi d'uso A e B, si dice che A include B se ogni istanza di A esegue B. B è incluso nell'esecuzione di A

e la responsabilità di esecuzione di B è unicamente di A. Un inclusione viene rappresentata con una freccia tratteggiata, che collega i casi d'uso coinvolti, in direzione del caso d'uso incluso;

- **Estensione:** L'estensione aumenta le funzionalità di uno use case. Dati due casi d'uso A e B, si dice che B estende A se A esegue B solo a determinate condizioni. L'esecuzione di B interrompe A e per questo motivo viene utilizzata prevalentemente per gestire errori ed eccezzioni. Un estensione viene rappresentata con una freccia tratteggiata, che collega i casi d'uso coinvolti, dal caso d'uso che estende a quello che viene esteso, e un quadrato con l'angolo in alto a destra piegato, contenente le condizioni necessarie per il verificarsi dell'estensione e il nome della stessa;

- **Generalizzazione:** La generalizzazione è un legame tra attori o più raramente tra use case. Dati due casi d'uso A e B, A è generalizzata di B se condivide almeno le funzionalità di A. B può modificare le funzionalità di A, mentre tutte le funzionalità non ridefinite si mantengono identiche a quelle di A. Le generalizzazioni vengono rappresentate con una freccia continua vuota dall'elemento figlio all'elemento padre.

**2.2.5.6.5 Diagrammi di sequenza:** I diagrammi di sequenza rappresentano dettagliatamente come gli oggetti interagiscono tra di loro tramite messaggi. Ogni entità del diagramma è collegata mediante una linea tratteggiata ad un'altra entità con lo stesso contenuto. Tale linea indica il passare del tempo. Le entità del diagramma si scambiano messaggi sotto forma di frecce che assumono una diversa struttura a seconda del tipo di messaggio che si sta inviando. I costrutti utilizzati in questi diagrammi sono i seguenti:

- **partecipante**: rappresenta un oggetto che detiene il flusso di esecuzione e collabora alla realizzazione di un comportamento.
  Il partecipante è così composto:
  - **nome**: nome dell'oggetto partecipante;
  - **barra di attivazione**: indica la durata del periodo di tempo durante il quale il partecipante è attivo;

- **messaggio**: rappresenta un'operazione di un partecipante che viene chiamata da parte di un altro partecipante e i dati scambiati tra i due.
  Un messaggio può essere:
  - **sincorno**: messaggio di chiamata in cui il partecipante chiamante attende la risposta del partecipante chiamato prima di proseguire la sua esecuzione. Viene utilizzata una freccia piena e sopra tale freccia va specificato il metodo invocato;

  - **asincrono**: messaggio di chiamata in cui il partecipante chiamante non attende la risposta del partecipante chiamato, ma prosegue la sua esecuzione subito dopo la chiamata. Viene utilizzata una freccia e sopra tale freccia va specificato il metodo invocato;
  - **ritorno**: messaggio di ritorno riferito ad un precedente messaggio di chiamata. Viene utilizzata una freccia tratteggiata e sopra tale freccia va indicato il tipo di ritorno;
  - **creazione**: messaggio di creazione di un nuovo partecipante da parte del partecipante chiamante. Viene utilizzata una freccia tratteggiata accompagnata dalla parola $<< create >>$;
  - **distruzione**: messaggio di distruzione di un partecipante da parte del partecipante chiamante. Viene utilizzata una freccia piena accompagnata dalla parola $<< destroy >>$.

- **frame di interazione**: rappresenta un ciclo o una condizione che coinvolge più messaggi e parti delle barre di attivazione di più partecipanti.
  Un frame è caratterizzato dalle proprietà di:
  - **guardia**: indica la condizione di attivazione del frame, posta in corrispondenza del partecipante coinvolto;
  - **etichetta**: indica la tipologia del frame.

Esistono diverse etichette per identificare i frame d'interazione. I progettisti dovranno attenersi alle seguenti:

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

## 2.2.6 Codifica

### 2.2.6.1 Scopo

La fase di codifica è la scrittura del codice per sviluppare la miglior soluzione del prodotto. In questa sezione si introdurranno tutte le norme necessarie allo sviluppo di un codice uniformato tra tutti i *Programmatori* e rispettoso delle regole standard indicate nel documento.

### 2.2.6.2 Aspettative

Conclusa la fase di codifica ci si attende un codice pulito e facile da leggere, utile nelle successive validazioni, modifiche e per agevolare la sua manutenzione. L'obiettivo è quello di sviluppare un prodotto conforme alle richieste individuate con il proponente$_G$.

### 2.2.6.3 Descrizione

In questa fase la programmazione del prodotto dovrà rispettare le norme descritte nel documento. Perseguendo gli obiettivi individuati nel documento *Piano di qualifica 2.0.0* si produrrà un software con un'alta qualità di codice.

### 2.2.6.4 Intestazione

Ciascun file di codifica dovrà riportare la seguente intestazione:

```
1 /*
2   Project Name: GDP- Gathering Detection Platform
3   File Name: fileName
4   Author: Name Surname
5   Creation Date: YYYY_MM_DD
6   Summary: file desctription
7   Last change date: YYYY_MM_DD
8 */
```

### 2.2.6.5 Stile di codifica

All'interno di questo paragrafo vengono elencate le norme da rispettare da ogni membro del gruppo per raggiungere uniformità del codice:

- **Indentazione**: ogni blocco di codice scritto per il prodotto da sviluppare deve essere ben indentato e deve rispettare per ciascun livello una misura di 4 spazi. E' obbligatorio utilizzare gli spazi anzichè le tabulazioni (tasto tab). Per riuscire a rispettare tale obbligo si cosiglia di configurare in modo appropriato il proprio editor o IDE. Fanno eccezione a queste regole i commenti che vengono inseriti per spiegazioni di blocchi di codice;
- **Parentesizzazione**: le parentesi devono essere utilizzate in linea col blocco di codice scritto e non in una linea sottostante, separandole con un singolo spazio;

- **Univocità delle classi, variabili, metodi e funzioni**: ogni classe, variabile, metodo e funzione utilizzata deve avere un nome significativo, esplicativo ed univoco;
- **Classi**: ogni classe deve avere il proprio nome scritto con l'iniziale maiuscola;
- **Metodi e funzioni**: il nome di metodi e funzioni devono iniziare per lettera minuscola e se composti da più parole le successive devono essere scritte con lettera maiuscola (stile *CamelCase*);
- **Commenti**: i commenti vanno inseriti solo nei casi in cui sono necessari per migliorare la comprensibilità e leggibilità del codice, mantenendo uno stile sintetico. Si possono utilizzare due tipologie di costrutti per i commenti:
  - `/* ... */` per i commenti che racchiudono più di una riga, lasciare la riga vuota dopo la dicitura /* ed andare a capo per descrivere una parte di codice e poi chiudere il commento con la dicitura */ nella riga successiva alla descrizione;
  - `// ...` per i commenti di linea singola che devono essere separati da uno spazio dopo la dicitura //.

  Questa dicitura legata ai commenti non è valida per il linguaggio Python$_G$ in quanto al loro posto si usa il cancelletto #. Il commento verrà scritto in linea al simbolo appena illustrato. I costrutti non devono essere riportati sulla stessa riga dell'istruzione a cui si riferiscono ma sulla riga che la precede;
- **Lingua**: il codice ed i commenti devono essere scritti in lingua inglese.

#### 2.2.6.6 Python

Si tratta di un linguaggio di programmazione definito "ad alto livello" rispetto alla maggior parte di essi. Si tratta di un linguaggio orientato ad oggetti, utile a sviluppare script, computazione numerica e sviluppare software. Nel progetto *Gathering-Detection-Platform*, Python$_G$ è il linguaggio su cui si basa tutto il backend$_G$, compreso il modulo del machine-learning$_G$.

- versione utilizzata: 3.8.x;
- link download: https://www.python.org/downloads/ .

#### 2.2.6.7 Java

Si tratta di una piattaforma che ha come caratteristica principale il fatto di rendere possibile scrittura ed esecuzione di applicazioni indipendenti dall'hardware di esecuzione. Il risultato è una virtualizzazione dalla piattaforma stessa, che rende così il linguaggio Java$_G$, e i relativi programmi, portabili su piattaforme hardware diverse.

- versione utilizzata: 11.x;
- link download: https://www.java.com/it/download/.

#### 2.2.6.8 HTML

HTML$_G$, acronimo di HyperText Markup Language, è un linguaggio di mark up per siti web. Era stato ideato per la formattazione e impaginazione di pagine ipertestuali sul web. Oggi giorno viene utilizzato soprattutto per gestire la separazione tra la struttura logica della pagina web e la sua rappresentazione, gestita dal CSS$_G$. Nel progetto questo linguaggio viene utilizzato per sviluppare la parte di web-app$_G$, interagendo con anche Java$_G$, CSS$_G$, Bootstrap$_G$ e Vue.js$_G$.

#### 2.2.6.9 CSS

Il CSS$_G$ è il principale linguaggio utilizzato per definire la formattazione dei siti e pagine web. L'utilizzo del CSS$_G$ permette di separare i contenuti della pagina HTML$_G$ dal proprio layout ma anche di rendere la programmazione più chiara e facile da utilizzare, garantendo il riutilizzo di codice e facilitando la manutenzione. Nel progetto viene utilizzato per formattare il layout estetico della web-app$_G$.

#### 2.2.6.10 Vue.js

È un framework JavaScript$_G$, configurato come Model-Control-View per la creazione di interfacce utente e applicazione single-page. Supporta molte funzionalità, anche avanzate, grazie ad una serie di librerie di supporto dedicate che sono ufficialmente mantenute.

- versione utilizzata: 2.6.12;
- link al sito: https://vuejs.org/.

### 2.2.7 Strumenti

#### 2.2.7.1 PragmaDB

Programma utilizzato per il tracciamento dei requisiti$_G$.

https://pragmadb.com/

#### 2.2.7.2 StarUML

Questo strumento viene usato per la realizzazione di diagrammi UML$_G$ in quanto è stato ritenuto semplice da utilizzare.

https://staruml.io/

#### 2.2.7.3 Atom

IDE che viene usato per la codifica del linguaggio Java$_G$ e Javascript$_G$, oltre a supportare anche altri molteplici linguaggi di programmazione come Python$_G$, C, C++ e anche LaTeX. Offre la piena compatibilità con Linux, Windows, macOS e fornisce molte integrazioni aggiuntive.

https://atom.io

#### 2.2.7.4 PyCharm

Si tratta di un IDE per programmare con il linguaggio Python$_G$. Offre molteplici plugin.

https://www.jetbrains.com/pycharm/

#### 2.2.7.5 Google.Colab

Colab, diminutivo di Colaboratory, è uno strumento online offerto da Google. Mette a disposizione l'hardware di Google in modo da permettere a chiunque di poter testare script o modelli pesanti (es. Machine Learning$_G$) nel caso la propria macchina non ne fosse in grado. Non prevede alcuna configurazione da parte dell'utente.

https://colab.research.google.com/notebooks/

#### 2.2.7.6 LeafLet

Utilizzato per la realizzazione delle Heat-map. Si tratta di un Open-Source basato su Javascript$_G$.

https://leafletjs.com/

#### 2.2.7.7 Maven

Maven$_G$ è uno strumento di build automation$_G$, sviluppato da *Apache*, utilizzato per la gestione di progetti Java$_G$. Maven$_G$ si basa sul concetto di *Project Object Model* (POM), ovvero un file .xml in cui sono specificate le informazioni e configurazioni necessarie allo sviluppo di un'applicazione Java. Maven$_G$, infatti, permette di configurare tutte le dipendenze ed i plugin, specificati nel pom.xml, autonomamente.

https://maven.apache.org/

#### 2.2.7.8 PostMan

È un'applicazione che consente di costruire, testare e documentare API più velocemente. Nel nostro caso è stato utilizzato per testare la connessione con il database e la correttezza delle query.

https://www.postman.com/

#### 2.2.7.9 Jupyter Notebook

Si tratta di un'applicazione web open-source$_G$ che ti permette di creare e condividere documenti che contengono codice, equazioni, visualizzazioni e testo narrativo, è particolarmente indicato per la pulizia dei dati e la loro trasformazione per l'utilizzo per il machine learning$_G$.

https://jupyter.org/index.html/

#### 2.2.7.10 Anaconda

Ambiente di distribuzione Python$_G$ che raccoglie molti pacchetti open source$_G$ e facili da installare, il vantaggio di usare questo strumento è la semplicità con cui si può creare il proprio ambiente di sviluppo.

https://www.anaconda.com/

# 3 Processi Di Supporto

I processi di supporto sono documentazione, gestione della configurazione, gestione della qualità, verifica e validazione.

## 3.1 Documentazione

### 3.1.1 Descrizione

Questa sezione fornisce le norme per la stesura, la verifica e l'approvazione dei documenti. Tali regole vanno seguite in tutti i documenti ufficiali prodotti durante il ciclo di vita del software, garantendo cosi la coerenza e la validità degli stessi.

### 3.1.2 Implementazione del documento

Per ogni documento che si intende sviluppare è necessario identificare:

- **titolo o nome:** che sia significativo ed ufficiale;
- **scopo:** che espliciti il contenuto generale del documento e la sua funzionalità come documentazione di progetto;
- **destinatari:** che indichi i soggetti a cui il documento è destinato, o coloro i quali sono tenuti a prenderne visione;
- **procedure$_a$ di gestione:** che guidino i responsabili nello sviluppo corretto e normato del documento, durante tutto il suo ciclo di vita;
- **versionamento:** pianificazione di versioni intermedie e finali del documento.

### 3.1.3 Ciclo di vita di un documento

Ogni documento prodotto percorre le tappe del seguente ciclo di vita:

- **creazione:** il documento viene creato partendo da un template progettato a tale scopo, situato nella cartella Template del repository$_G$ remoto;
- **strutturazione:** il documento viene fornito di un registro delle modifiche, di un indice dei contenuti e, se necessario, di un indice delle figure e di un indice delle tabelle presenti nel corpo del documento;

- **stesura:** il corpo del documento viene scritto progressivamente, da più membri del gruppo, adottando un metodo incrementale;
- **revisione:** ogni singola sezione del corpo del documento viene regolarmente rivista da almeno un membro del gruppo, che deve essere obbligatoriamente diverso dal redattore della parte in verifica; se necessario, la verifica può essere svolta da più persone: in questo caso può partecipare anche chi ha scritto la sezione in verifica a patto che non si occupi della parte da esso redatta;
- **approvazione:** terminata la revisione, il *Responsabile di Progetto* stabilisce la validità del documento, che solo a questo punto può essere considerato completo e può essere quindi rilasciato.

### 3.1.4 Template in formato LaTeX

Il gruppo ha deciso di adottare il linguaggio LATEX per la stesura dei documenti. E' stato definito un template per automatizzare l'applicazione delle norme tipografiche e di formattazione, in funzione della coerenza e coesione dei prodotti finali. L'uso di un template comune per la strutturazione dei documenti, inoltre, permette di rendere più efficiente l'applicazione di nuove norme o di modifiche a norme già esistenti a tutti i documenti redatti fino a quel momento.

### 3.1.5 Documenti prodotti

**Formali:** sono i documenti che riportano le norme che regolano l'operato del gruppo e gli esiti delle attività$_G$ da esso portate avanti nel corso del ciclo di vita del software. Le caratteristiche di un documento formale sono:

- storicizzazione delle versione del documento prodotte durante la sua stesura;
- attribuzione di nomi univoci ad ogni versione;
- approvazione della versione definitiva da parte del *Responsabile di Progetto*.

Se un documento formale ha più versioni, si considera come corrente sempre la più recente tra quelle approvate dal *Responsabile di Progetto*. I documenti formali possono essere classificati come Interni o Esterni:

- **interni:** che riguardano le dinamiche interne del gruppo, di marginale interesse per committenti$_G$ e proponente$_G$;
- **esterni:** che interessano i committenti$_G$ ed il proponente$_G$ e che vengono loro consegnati nell'ultima versione approvata.

Di seguito sono elencati i documenti ufficiali prodotti e la loro classificazione in uso Interno o Esterno:

- **norme di progetto**: documento ad uso Interno. Contiene le norme e le regole, stabilite dei membri del gruppo, alle quali ci si dovrà attenere durante l intera durata del lavoro di progetto;
- **glossario**: documento ad uso Esterno. Elenco ordinato di tutti i termini usati nella documentazione che il gruppo ritiene necessitino di una definizione esplicita, al fine di rimuovere ogni ambiguità;
- **studio di fattibilità:** documento ad uso Interno. Lo Studio di Fattibilità ha l'obiettivo di esporre (brevemente) ogni capitolato$_G$ e di elencare per ognuno gli aspetti positivi e le criticitaà che il team ha individuato;
- **piano di progetto**: documento ad uso Esterno. Lo scopo del Piano di Progetto è organizzare le attività$_G$ in modo da gestire le risorse disponibili in termini di tempo e "forza lavoro";
- **piano di qualifica**: documento ad uso Esterno. Lo scopo del Piano di Qualifica è presentare i metodi di verifica e validazione implementati dal gruppo, per garantire la qualità del capitolato$_G$ scelto;
- **analisi dei requisiti$_G$**: documento ad uso Esterno. Lo scopo dell'analisi dei Requisiti è esporre dettagliatamente i requisiti$_G$ individuati per lo sviluppo del capitolato$_G$ scelto.

**Informali:** un documento è informale se:

- non è stato ancora approvato dal *Responsabile di Progetto*;
- non è soggetto a versionamento.

I documenti appartenenti alla seconda categoria saranno i verbali, che potranno essere:

- **interni:** resoconti sintetici degli incontri dei membri del gruppo, contengono un ordine del giorno, riportano gli argomenti affrontati e le decisioni prese;
- **esterni:** rapporti degli incontri del gruppo coi committenti$_G$ e/o col proponente$_G$, strutturati secondo lo schema domanda-risposta.

Per i verbali è prevista un'unica stesura. Tale scelta è dettata dal fatto che apportarvi modifiche significherebbe cambiare le decisioni prese in sede di riunione.

### 3.1.6 Directory di un documento

Ogni documento è racchiuso all'interno di una directory che prende il nome dal documento ivi trattato; essa è posizionata a sua volta all'interno della directory **Documenti Interni** o **Documenti Esterni**, a seconda della tipologia del documento. Quest'ultima, il file TEX principale e il documento pdf da esso generato adottano la convenzione *Snake case*, come stabilito nella sottosezione 3.1.8; nel caso il documento sia formale, in coda al suo nome appare anche la sua versione (e.g. *norme_di_progetto_1.0.0*). Tutti i capitoli appartenenti ad un documento sono organizzati in una subdirectory **capitoli** posta allo stesso livello del file TEX principale.

### 3.1.7 Struttura generale dei documenti

#### 3.1.7.1 Frontespizio

Il frontespizio è la prima pagina di ogni documento. La prima pagina di ogni documento sarà composta da:

- **logo del gruppo**;
- **indirizzo e-mail del gruppo**;
- **nome del gruppo**.

Informazioni sul documento che includono:

- **versione**: indica la versione attuale del documento;
- **approvazione**: indica chi ha approvato il documento;
- **redazione**: indica la lista dei redattori del documento;
- **verifica**: indica la lista dei verificatori del documento;
- **stato**: indica lo stato attuale in cui si trova il documento;
- **uso**: indica l'uso finale del documento (interno o esterno);
- **sommario:** posto in fondo alla pagina che contiene una breve descrizione del contenuto del documento.

#### 3.1.7.2 Registro Modifiche

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

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

#### 3.1.7.3 Indice

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

#### 3.1.7.4 Corpo del documento

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

- in alto a sinistra è presente il logo del gruppo;
- in alto a destra è riportata la sezione alla quale la pagina appartiene;
- il contenuto principale è posto tra l'intestazione e il piè di pagina;
- una riga divide il contenuto principale e il piè di pagina;
- in basso è riportato il numero di pagina attuale ed il numero totale delle pagine che compongono il documento.

#### 3.1.7.5 Verbali

I verbali vengono identificati attraverso da un codice alfanumerico:

**v_[Tipo_verbale]_[Data]**

La definizione di questa classificazione è la seguente:

- **v**: indica verbale;
- **Tipo_verbale**: classifica il verbale in:
  - **i**: indicazione per il verbale interno;
  - **e**: indicazione per il verbale esterno.
- **Data**: indica la data nel formato *YYYY_MM_DD* del giorno della riunione.

Ai verbali, sia interni che esterni, si applicano le stesse norme strutturali degli altri documenti. Il contenuto di un verbale è così organizzato:

- **luogo:** riporta il luogo in cui si è svolta la riunione (in alternativa il mezzo utilizzato es. Discord$_G$);
- **data:** riporta la data della riunione;
- **ora di inizio:** riporta l'ora in cui è iniziata la riunione;
- **ora di fine:** riporta l'ora in cui è terminata la riunione;
- **partecipanti:** riporta l'elenco dei presenti alla riunione;
- **ordine del giorno:** contiene l'elenco degli argomenti affrontati alla riunione;
- **decisioni:** contiene le decisioni prese durante la riunione, in forma di lista individuati da un codice identificativo per ogni decisione.

I codici di identificazione delle decisioni sono descritti di seguito:

**[tipoVerbale]_[dataVerbale].X**

La definizione della classificazione è la seguente:

- **tipoVerbale**: individuato dalle vocali:
  - **I**: per verbale interno;
  - **E**: per verbale esterno.
- **dataVerbale**: individua la data del verbale;
- **X**: numero incrementale con inizio da 1.

### 3.1.8 Norme Tipografiche

Per attribuire uniformità e coerenza alla documentazione sono state concordate delle norme tipografiche da adottare durante tutta la sua stesura, esposte nelle seguenti sezioni.

#### 3.1.8.1 Convenzioni di denominazione

I nomi delle directory e dei documenti prodotti rispettano la convenzione *Snake case*:

- i nomi fanno utilizzo esclusivo del minuscolo;
- nel caso il nome sia composto da più parole, è necessaria la presenza del carattere separatore *underscore* "_";
- non è prevista l'omissione delle preposizioni.

Le estensione dei file sono ovviamente escluse da questa convenzione.

#### 3.1.8.2 Termini del Glossario

Ogni termine del *Glossario* è contrassegnato, in ogni sua istanza, da una "G" maiuscola a pedice; la prima occorrenza di un termine all'interno di un documento presenta una "G" di dimensione standard, mentre le successive "G" (all'interno dello stesso documento) sono di dimensione ridotta per non risultare eccessivamente intrusive ed ostacolare la lettura. Le istanze dei termini del Glossario presenti nei titoli non necessitano la lettera a pedice.

#### 3.1.8.3 Formato di data

Le date rispettano il formato [YYYY]-[MM]-[DD] dove:

- **YYYY:** corrisponde all'anno;
- **MM:** corrisponde al mese;
- **DD:** corrisponde al giorno.

#### 3.1.8.4 Sigle

Per ragioni di scorrevolezza e brevità sono presenti nella documentazione alcune abbreviazioni di parole ricorrenti, elencate in seguito organizzate per categorie.
Revisioni:

- **RR:** revisione dei requisiti$_G$;
- **RP:** revisione di progettazione;

- **RQ:** revisione di qualifica;
- **RA:** revisione di accettazione.

Documentazione Interna ed Esterna:

- **AdR:** analisi dei requisiti$_o$;
- **NdP:** norme di progetto;
- **PdQ:** piano di qualifica;
- **PdP:** piano di progetto;
- **MU:** manuale utente;
- **MS:** manuale sviluppatore;
- **G:** glossario;
- **V:** verbale.

Ruoli di progetto:

- **Re:** responsabile;
- **Am:** amministratore;
- **An:** analista;
- **Pgt:** progettista;
- **Pgr:** programmatore;
- **Ve:** verificatore.

### 3.1.9 Elementi grafici

#### 3.1.9.1 Immagini

Questa sezione definisce le norme per l'uso di elementi grafici quali immagini, tabelle e diagrammi. Le immagini apportano un valore aggiunto alla descrizione o forniscono una rappresentazione grafica di ciò che si sta presentando. Immagini con funzione puramente estetica non sono pertanto ammesse, ad eccezione di quanto definito nel template comune. Tutte le immagini sono centrate all'interno della pagina e munite di una breve didascalia così formata:

**Figura X: breve descrizione dell'immagine**

dove X indica la numerazione dell'immagine.

#### 3.1.9.2 Grafici

I grafici in linguaggio UML, usati per la modellazione dei casi d'uso e per i diagrammi della progettazione, sono inseriti come immagini.

#### 3.1.9.3 Tabelle

L'uso di tabelle è consigliato solo quando strettamente necessario. La rappresentazione dei dati in forma tabellare è obbligatoria solo nel momento in cui risulti molto difficile organizzare informazioni aventi una struttura complessa. È obbligatorio l'uso di colori che abbiano un contrasto sufficiente a garantire la leggibilità. Le tabelle eccessivamente lunghe sono sconsigliate, poichè potrebbero risultare dispersive.

### 3.1.10 Metriche

#### 3.1.10.1 MQPD02 Indice Gulpease

L'indice di Gulpease$_G$ riporta il grado di leggibilità di un testo redatto in lingua italiana.
La formula adottata è:

$$\text{GULP}= 89+ \frac{300*(numerofrasi)-10*(numeroparole)}{numerolettere}$$

L'indice così calcolato può pertanto assumere valori compresi tra 0 e 100, in cui:

- **GULP< 80:** indica una leggibilità difficile per un utente con licenza elementare;
- **GULP< 60:** indica una leggibilità difficile per un utente con licenza media;
- **GULP< 40:** indica una leggibilità difficile per un utente con licenza superiore.

#### 3.1.10.2 Correttezza Ortografica

La correttezza ortografica della lingua italiana è verificata attraverso l'apposito strumento integrato di TEXstudio, il quale sottolinea in tempo reale le parole ove ritiene sia presente un errore, consentendone la correzione.

### 3.1.11 Strumenti di stesura

#### 3.1.11.1 Latex

Per la stesura dei documenti, il gruppo *Jawa Druids* ha scelto LATEX, un linguaggio compilato basato sul programma di composizione tipografica TEX, al fine di produrre documenti coerenti, ordinati, templatizzati e stesi in modo collaborativo.

## 3.2 Gestione della configurazione

### 3.2.1 Scopo

Lo scopo della configurazione è definire una precisa organizzazione nella produzione di documentazione e codice. L'implementazione di questo processo rende sistematica la produzione di codice e documenti, la loro modifica e il loro avanzamento di stato. Ogni elemento relativo al progetto garantisce pertanto il suo versionamento e rispetta le norme di collocazione, denominazione, modifica e assegnazione di stato descritte in seguito. Sono inoltre qui raggruppati e brevemente descritti gli strumenti utilizzati a supporto di tale organizzazione.

### 3.2.2 Versionamento

#### 3.2.2.1 Codice di versione di un documento

Ogni documento possiede una storia che dev'essere ricostruibile tramite i suoi codici di versione. Il registro delle modifiche, presente in ogni documento, raccoglie tutta la storia delle versioni con le modifiche ad esse associate. Ogni versione corrisponde ad una riga in tale registro ed è composta da tre numeri che possono assumere valori interi, con incremento di una singola unità alla volta. Questi tre valori sono separati da punti nel seguente modo:

**X.Y.Z**

ove, partendo dall'ultima lettera:

- **Z** rappresenta una versione in via di sviluppo del documento, ovvero una versione in cui i redattori hanno aggiunto dei nuovi capitoli e/o sezioni o modificati quelli preesistenti, con relativa verifica. La numerazione di tale indice comincia da 0 e viene incrementato ad ogni modifica, ripartendo dal valore di 0 ogni volta in cui viene incrementato l'indice **X** o **Y**;
- **Y** rappresenta una versione in cui uno o più *Verificatori* hanno proceduto alla revisione del documento, assicurandone l'omogeneità, la compattezza e la correttezza sia grammaticale che strutturale. La numerazione di tale indice comincia da 0, ripartendo da tale valore ogni volta in cui viene incrementato l'indice **X**;
- **X** rappresenta una versione ufficiale approvata dal *Responsabile di progetto*, e pertanto garantisce un particolare livello di stabilità, correttezza e professionalità. La numerazione di tale indice comincia da 0 e non retrocede mai a questo valore.

#### 3.2.2.2 Tecnologie adottate

Il gruppo utilizza il sistema di controllo di versione Git$_G$ con hosting sulla piattaforma Github$_G$. L'interazione con queste tecnologie avviene da linea di comando del terminale attraverso il wrapper$_G$ Git-flow$_G$ oppure tramite il software ad interfaccia grafica GitKraken$_G$. L'utilizzo di questi strumenti assicurano la progressione, collaborazione e sicurezza nello sviluppo di ogni file all'interno della repository$_G$.

#### 3.2.2.3 Repository remoto

Il repository$_G$ remoto utilizzato dal gruppo è disponibile al link

https://github.com/Andrea-Dorigo/jawadruids.git

È possibile scaricare l'intero progetto sulla propria macchina tramite il comando

```
git clone https://github.com/Andrea-Dorigo/jawadruids.git
```

La struttura dei branch rispetta la convenzione standard comunemente accettata dalla community di Git$_G$ e Git-flow$_G$. Il branch `main` contiene la versione ufficiale del progetto, in cui il *Responsabile* ha approvato tutti i files in esso contenuti. Da questo si dirama il branch `develop`, il quale contiene files nella maggior parte dei casi già revisionati dai *Verificatori*; fanno eccezione a questa norma$_G$ i documenti e file di interesse comune a moltepici ambiti del progetto oppure i file che non richiedono particolare verifica (gitignore, template). A partire dal `develop` si diramano i branch delle `feature`, `bugfix` e `hotfix`; i loro nomi devono esplicitare ciò che si sta producendo al loro interno, sempre rispettando le convenzioni di Git-flow$_G$. La cartella **documentazione** contiene tutti i documenti prodotti dal gruppo; le norme riguardanti i suoi contenuti si trovano nella sezione § 3.1.6.

## 3.3 Gestione della qualità

### 3.3.1 Scopo

Lo scopo della gestione della qualità è assicurare il rispetto degli obiettivi di qualità da parte del prodotto finale ed il soddisfacimento delle esigenze manifestate dal proponente$_G$.

### 3.3.2 Descrizione

Il documento Piano di Qualifica tratta in maniera più approfondita della gestione della qualità. In particolare descrive le modalità e le metriche utilizzate per valutare, e garantire, la qualità di processo e di prodotto. Inoltre riporta i test da applicare per verificare l'adempimento dei requisiti$_G$ del prodotto software. Tale documento si concentra sui seguenti punti:

- Presentazione degli standard a cui si fa riferimento;
- Identificazione degli attributi significativi per il prodotto software;
- Identificazione dei processi interessati negli standard adottati.

Per ogni processo si descrivono:

- Obiettivi da perseguire;
- Strategie;
- Metriche da applicare.

Per ogni prodotto si descrivono:

- Metriche da applicare.

### 3.3.3 Aspettative

Le aspettative che si intendono soddisfare con la gestione della qualità sono le seguenti:

- Raggiungimento della qualità di processo;
- Raggiungimento della qualità di prodotto;
- Raggiungimento della qualità dell'organizzazione del gruppo;
- Prova oggettiva della qualità;
- Ottenimento della piena soddisfazione finale del proponente$_G$ e del cliente.

### 3.3.4 Attività

Il processo della gestione della qualità presenta le seguenti attività$_G$:

- **Pianificazione:** delineare gli obiettivi di qualità e le metriche e strategie per poterli raggiungere, approntare le risorse a disposizione per svolgere le attività$_G$ nel migliore dei modi;
- **Esecuzione:** applicare quanto stabilito;
- **Controllo:** misurare e monitorare i risultati ottenuti;
- **Aggiustamento:** a fronte dei risultati, è necessario adeguare le strategie e le metriche, individuando, inoltre, dove effettuare dei miglioramenti.

### 3.3.5 Denominazione delle metriche

Ogni metrica possiede un codice univoco che la identifica. Tale codice ha il seguente modello:

**MQ[Categoria][Numero]**

dove:

- **M** specifica che ci si sta riferendo ad una metrica;
- **Q** specifica che si tratta di una metrica riguardo la qualità;
- [**Categoria**] specifica a quale categoria appartiene la metrica; le categorie sono:
  - **PS** per i processi;
  - **PD** per i prodotti.
- [**Numero**] specifica l'identificativo numerico specifico per ogni metrica. Il conteggio inizia da 1.

## 3.4 Verifica

### 3.4.1 Descrizione

Il processo di verifica prende in input ciò che è già stato prodotto, ad esempio una versione di un documento, e lo restituisce in uno stato conforme alle aspettative. Per ottenere tale risultato ci si affida a processi di analisi e test.

### 3.4.2 Scopo

L'obiettivo di tale processo è di avere una sola versione dei prodotti realizzati e che essi siano corretti, coesi e completi. Sono soggetti a verifica sia il software che la documentazione.

### 3.4.3 Aspettative

Il corretto svolgimento del processo di verifica avviene attraverso i seguenti punti:

- viene effettuato seguendo delle procedure$_G$ già definite;
- la verifica segue dei criteri chiari ed affidabili;
- il prodotto viene verificato nel corso di ognuna delle sue fasi;
- al termine della verifica il prodotto si trova in uno stato stabile;
- una volta verificato, possiamo procedere con la fase di validazione.

### 3.4.4 Attività

#### 3.4.4.1 Analisi

Il processo di analisi si suddivide in analisi statica ed analisi dinamica.

##### 3.4.4.1.1 Analisi statica

L'analisi statica effettua controlli su documenti e codice, di cui valuta e applica la correttezza(intesa come assenza di errori e difetti), la conformità a regole e la coesione$_G$ dei componenti. Per effettuare analisi statica esistono metodi manuali di lettura(attuati da persone) e metodi formali ed automatizzati(attuati da macchine).
Quelli manuali sono due:

- **Walkthrough$_G$:** i vari componenti del team analizzano gli oggetti nella loro totalità per cercare anomalie, senza sapere inizialmente se vi siano difetti, quali e dove siano;
- **Inspection$_G$:** i verificatori utilizzano delle liste di controllo per fare ispezione cercando errori specifici in sezioni specifiche.

##### 3.4.4.1.2 Analisi dinamica

L'analisi dinamica è una tecnica di analisi del prodotto software che richiede la sua esecuzione. Viene effettuata mediante dei test che verificano se il prodotto funziona e se ci sono anomalie.

#### 3.4.4.2 Test

I test sono una parte fondamentale del processo di verifica: producono una misura della qualità del prodotto. Questi hanno lo scopo di verificare che il prodotto software svolga le attività$_G$ che sono state richieste dal proponente$_G$, e di permettere di individuare e rimuovere gli errori commessi, prima di mettere il prodotto in uso. In questo modo si aumenta la qualità del software. I test devono essere:

- **Automatici:** si intende la presenza di un automazione che permette ad ogni membro del gruppo di invocare ed eseguire tutti o una parte dei test. Quest'ultimi devono essere semplici e rapidi e soprattutto non devono richiedere l'interazione umana;
- **Ripetibili:** si intende che ogni invocazione di un test deve generare sempre lo stesso risultato dato uno stesso insieme di dati in input. Per assicurare ciò serve il determinismo, che è garantito se:
  - Non cambia stato iniziale del sistema;
  - Non cambia ambiente di esecuzione.

Per ogni test si specificano i seguenti parametri:

- **Ambiente:** sistema hardware e software usato nel corso del test;
- **Stato iniziale:** stato di partenza del prodotto software al momento del test;
- **Input:** dati in ingresso;
- **Output:** dati in uscita attesi;
- **Avvisi addizionali:** eventuali ulteriori istruzioni su come eseguire il test e su come interpretare i risultati ottenuti.

I tipi di test sviluppati sono i seguenti:

**Test di unità**
I test di unità, detti anche unit test, hanno il compito$_G$ di testare le singole unità software, dove per unità si intende il più piccolo componente del programma dotato di funzionamento autonomo. Sono utilizzati prettamente dagli sviluppatori e possono essere sia manuali che automatici. Questi test facilitano le modifiche del codice del modulo in momenti successivi, infatti scrivendo test case per ogni funzione e metodo, in caso di fallimento diventa facile trovare l'errore. Vengono eseguiti sempre prima dell'integrazione del modulo nel software in quanto garantiscono automaticamente l'integrità del codice. I test vengono fatti mediante driver e software che simulano chiamate da parte di utenti o del sistema.

**Test di integrazione**
I test di integrazione servono a verificare le corrette funzionalità delle interfacce rispetto al prodotto software. Vengono solitamente usate due o più unità già testate in modo da poter integrare i moduli successivi, in caso di successo del test, per poi espandersi iniziando ad unire anche tutte le parti software restanti.

**Test di sistema**
Questa tipologia di test serve a verificare il corretto funzionamento del prodotto una volta integrato completamente, per verificare che i requisiti$_a$ specificati nell'*Analisi dei Requisiti* funzionino.

**Test di regressione**
Questa tipologia di test ha lo scopo di testare le nuove funzionalità e garantire che le funzionalità preesistenti abbiano mantenuto le loro caratteristiche qualitative dopo l'introduzione di queste ultime. Per questo è necessario rieseguire i test già esistenti sul codice modificato così da verificare se il comportamento degli elementi precedentemente funzionanti si è alterato per via delle modifiche introdotte.

**Test di accettazione**
Ultima tipologia di test, vengono fatti per verificare il corretto funzionamento del prodotto software una volta completato. Oltre agli sviluppatori, questi test vengono eseguiti sotto l'occhio del proponente$_G$ cosicché possano valutare se il prodotto soddisfi i requisiti$_G$ richiesti nel capitolato d'appalto$_G$. Se i test vengono superati significa che il software è pronto per il rilascio.

#### 3.4.4.3 Identificazione dei test

I test presenti nella sezione precedente sono descritti così:

- **Id Test**;
- **Descrizione**;
- **Esito**:
  - **NI**: non implementato;
  - **I**: implementato;
  - **NS**: non superato;
  - **S**: superato.

Ogni test sui requisiti$_G$, ovvero test di *sistema* e *accettazione*, ha questa nomenclatura:

**[TipoTest]RS[classificazione][tipo_di_requisito][codice_requisito]**

In cui:

- **TipoTest**: specifica il test che si sta facendo;
- **RS**: specifica che il test è fatto su un requisito$_G$;
- **Classificazione**: individua la classificazione del requisito$_G$:
  - **F**: indica se il requisito$_G$ è funzionale;
  - **Q**: indica se il requisito$_G$ è qualitativo;
  - **V**: indica se il requisito$_G$ è vincolante;
  - **P**: indica se il requisito$_G$ è prestazionale.
- **tipo_di_requisito**: suddiviso in:
  - **O**: se il requisito$_G$ è obbligatorio;
  - **D**: se il requisito$_G$ è desiderabile;

- **F**: se il requisito$_G$ è facoltativo.

- **codice_requisito**: si tratta di un numero incrementale per rendere univoco il requisito$_G$.

Invece i test di *unità*, *integrazione* e *regressione*, sono rappresentati nel seguente modo:

**[TipoTest][Id]**

Dove:

- **Id**: rappresenta un numero incrementale partendo da 1.

### 3.4.5 Strumenti

#### 3.4.5.1 Analisi statica

- **Checkstyle$_G$:** è uno strumento che permette di eseguire l'analisi statica del codice utilizzato nello sviluppo di un progetto software, per verificare se il codice sorgente Java$_G$ è conforme alle regole di codifica adottate dal gruppo. Tali regole sono quindi definite all'interno del file *checkstyle.xml*, il quale viene integrato dal gestore delle dipendenze di *Maven*$_G$. L'uso di tale strumento permette di automatizzare il processo di controllo del codice Java$_G$: viene eseguito ad ogni build, segnalando gli eventuali errori riscontrati;

- **Pylint$_G$:** è uno strumento utilizzato per l'analisi statica del codice sorgente Python$_G$. Viene quindi adottato per controllare la presenza di errori nel codice, con l'obiettivo di applicare uno standard codifica e di promuovere buone prassi di scrittura del codice;

- **Prettier$_G$:** è uno strumento per il controllo automatico della formattazione del codice scritto in linguaggio JavaScript$_G$, esso permette di eseguire dei controlli sul formato del codice tramite regole configurabili. Oltre ad effettuare il controllo, mette a disposizione anche una funzionalità di formattazione del codice attivabile al momento del salvataggio.

- **ESLint$_G$:** è uno strumento di analisi statica del codice per identificare le problematiche trovate nel codice JavaScript$_G$. Permette di eseguire controlli sia per quanto riguarda la qualità del codice che per lo stile di codifica.

#### 3.4.5.2 Analisi Dinamica

- **JUnit$_G$:** è un framework$_G$, integrato dal gestore di dipendenza Maven$_G$, che permette di eseguire automaticamente i test di unità per codice Java$_G$. Permette di essere configurato direttamente all'interno del file *pom.xml* presente nella cartella di progetto e, ad ogni build del prodotto software, esegue automaticamente tutti i test definiti nel progetto, segnalando quelli che non hanno avuto esito positivo, attraverso opportuni errori. Non necessita di particolari configurazioni in quanto autonomamente rileva i test disponibili, purchè rispettino le convenzioni di nomenclatura ed utilizzino opportunamente i metodi e le annotazioni definite dal framework$_G$ per i test di unità.

### 3.4.6 Metriche

#### 3.4.6.1 MQPD03 Rilevamento Errori

Indice che mostra qual è la percentuale di errore basata sui test fatti. Come formula viene utilizzata la seguente:

$$\textbf{RE} = (1 - \frac{TE}{TT})\textbf{*100}$$

- **RE** sta per *Rilevamento Errori*;
- **TE** sta per *Test con Errori*;
- **TT** sta per *Test Totali.*

*Il gruppo Jawa Druids ha valutato precoce la scelta di stabilire in questa prima fase dei valori soglia per tale metrica, di conseguenza il gruppo si riserva di integrarli successivamente in modo opportuno.*

#### 3.4.6.2 MQPD04 Validità Dati

Questo indice misura la effettiva veridicità dei dati che arrivano in input al software. Ovviamente più i dati si avvicinano alla realtà più elevato sarà il valore dell'indice. Viene utilizzata la seguente formula:

$$\textbf{VD} = (\ \frac{DIV}{DP}\ )\ \textbf{*100}$$

- **VD** sta per *Validità Dati*;
- **DIV** sta per *Dati Input Validati*;
- **DP** sta per *Dati Previsti.*

*I range di valori accettabili non si possono ancora esprimere in quanto, concordi con l'azienda, si stabiliranno in futuro.*

#### 3.4.6.3 MQPD05 Comprensione del codice

Questo indice viene calcolato, mediante percentuale, per rappresentare la facilità di comprensione dell'utente riguardo il codice. L'operazione è costituita dalla divisione tra le *linee di commento* e le *linee di codice*:

$$\textbf{F} = (\frac{N_{Lc}}{N_{Lcod}})\textbf{*100}$$

- **F** sta per *Facilità*, ovvero la facilità di comprensione del codice;
- **N_Lc** sta per *Numero di linee di commento*;
- **N_Lcod** sta per *Numero di linee di codice.*

#### 3.4.6.4 MQPD06 Structural fan-in

Indice che misura il numero di moduli che utilizzano un dato modulo. Un elevato valore di tale indice significa un alto uso del modulo in questione.

## 3.5 Validazione

### 3.5.1 Scopo

La validazione ha come scopo finale quello di garantire che il software rispetti i requisiti$_G$ posti e soddisfi le esigenze del proponente$_G$.

### 3.5.2 Descrizione

Il processo di validazione esamina il prodotto della fase di verifica ed accerta che questo sia pienamente conforme alle attese e ai requisiti$_G$ del proponente$_G$.

### 3.5.3 Aspettative

La validazione ha dei criteri da rispettare:

- identificazione degli oggetti da validare;
- identificazione di una strategia avente procedure$_G$ per la validazione in cui i test di verifica possono essere riutilizzati;
- applicazione di tale strategia;
- verificare la coerenza dei risultati ottenuti con quelli aspettati.

*Questa sezione sarà opportunamente integrata quando bisognerà normare la validazione del codice.*

# 4 Processi Organizzativi

## 4.1 Processo di coordinamento

### 4.1.1 Scopo

Questa sezione mostra i metodi di coordinamento adottati dal gruppo *Jawa Druids* in termini di riunioni, comunicazione, ruoli del progetto e assegnazione dei compiti. Verranno inoltre brevemente introdotti gli strumenti selezionati e la loro configurazione di base. La struttura del processo di coordinamento secondo lo standard ISO/ IEC 12207 è la seguente:

- **Comunicazione:** interna oppure esterna;
- **Riunioni:** interne oppure esterne.

### 4.1.2 Comunicazione

Durante il progetto, il team di *Jawa Druids* comunicherà su due diversi livelli: interno ed esterno. Per quanto riguarda la comunicazione esterna, esse avverranno con i seguenti soggetti:

- **Proponente**$_G$**:** l'azienda *Sync Lab*, rappresentata dal signor Fabio Pallaro;
- **Committenti**$_G$**:** nella persona del prof. Tullio Vardanega e del prof. Riccardo Cardin;
- **Competitor:** questo punto verrà chiarito dopo la Revisione dei Requisiti$_G$ quando i capitolati saranno assegnati ai relativi gruppi;
- **Esperti interni:** da consultare eventualmente previo accordo con il proponente$_G$ ed i committenti$_G$.

Il gruppo si rivolgerà a tutti i soggetti mediante comunicazioni scritte e/o meeting.

#### 4.1.2.1 Comunicazione interna

Il metodo di comunicazione standard per l'interazione scritta tra i membri del gruppo *Jawa Druids* è il servizio di messaggistica istantanea Discord$_G$. La strategia per la gestione delle discussioni sarà anche creare un canale specifico per ogni attività$_G$, non ignorabile, che il gruppo deve scegliere (ad esempio, tutte le discussioni sui documenti saranno condotte all'interno di quel canale specifico). Oltre a questa tipologia di canali, le discussioni saranno suddivise anche nelle seguenti categorie:

- **git-github:** per qualsiasi discussione e/o problemi riguardanti la repository$_G$;

- **generale:** per qualsiasi discussione generica o riguardante il progetto;
- **LaTeX:** per qualsiasi discussione riguardante LaTeX;
- **domande per azienda:** per tutte le domande/ dubbi da porre al proponente$_G$.

Discord$_G$ permette di notificare un particolare messaggio ad una o più persone, includendo nel corpo del messaggio le seguenti parole chiave:

- **@everyone:** il messaggio verrà notificato a tutti i componenti del gruppo;
- **@username:** inserendo l'username specifico, il messaggio verrà notificato all'utente desiderato.

Inoltre, un'altro modo di comunicazione è la video-chiamata di Discord$_G$, che è stato preferito ad altri servizi per la sua versatilità, per il fatto di essere open-source$_G$ e per una essere multi-piattaforma.

#### 4.1.2.2 Comunicazione esterna

Il Responsabile di Progetto rappresenterà l'intero team e manterrà i contatti esterni tramite un indirizzo email appositamente creato. L'e-mail utilizzata sarà:

**JawaDruids@gmail.com**

Il Responsabile di Progetto deve utilizzare regole relative alle comunicazioni interne per notificare a ciascun membro del team eventuali comunicazioni ricevute dal cliente e dal proponente$_G$. Ogni email inviata avrà la seguente struttura:

- **Oggetto:** l'oggetto della mail sarà preceduto dalla sigla "[SWE][UNIPD]", in modo che l'ambito di riferimento dell'e-mail sia immediatamente chiaro ed espresso nel modo più conciso possibile per eliminare ambiguità e migliorare la comprensione dell'oggetto;
- **Corpo:** il tono da mantenere è formale, ci si rivolgerà dando del Lei. Il corpo sarà sempre preceduto da una formula di apertura formale, come "Egregio", "Alla cortese attenzione di *Sync Lab*", "Spettabile", a seconda del destinatario. Il contenuto dovrà inoltre essere sintetico ed esaustivo, per esporre al meglio il problema e/o eventuali richieste.

### 4.1.3 Riunioni

Ogni riunione nominerà un segretario il cui compito$_G$ è tenere traccia di ciò che viene discusso, eseguire l'ordine del giorno e infine utilizzare le informazioni raccolte per redigere i verbali della riunione.

#### 4.1.3.1 Riunioni interne

Solo i membri del team possono partecipare alle riunioni interne. Almeno quattro persone devono partecipare alla riunione per confermarne la validità. Le decisioni saranno prese a maggioranza semplice ed inoltre, affinché la riunione sia considerata efficace, il responsabile del progetto deve completare le seguenti attività$_G$:

- fissare preventivamente la data della riunione, previa verifica della disponibilità dei membri del gruppo;
- stabilire un ordine del giorno;
- nominare un segretario per la riunione.

I partecipanti della riunione, invece, devono:

- avvertire preventivamente in caso di assenze o ritardi;
- presentarsi puntuali al meeting nell'ora prestabilita;
- partecipare attivamente alla discussione.

Ogni membro del gruppo avrà la possibilità di richiedere un incontro interno.

#### 4.1.3.2 Riunioni esterne

Le riunioni esterne prevedono la partecipazione di soggetti esterni, quali il proponente$_G$ e i committenti$_G$, oltre ai componenti del gruppo *Jawa Druids*. Così come le riunioni interne, anche quelle esterne, prevedono la nomina di un segretario che dovrà poi redigere un verbale di riunione. Le riunioni esterne con il proponente$_G$ saranno condotte attraverso il canale Discord$_G$ creato appositamente.

### 4.1.4 Strumenti utilizzati per il processo di coordinamento

- **Discord$_G$:** https://discordapp.com/company/;
- **Gmail**: https://www.google.com/intl/it/gmail/about/.

## 4.2 Processo di pianificazione

### 4.2.1 Scopo

Lo scopo di questa sezione è spiegare come il gruppo intende pianificare il lavoro, dall'assegnazione dei ruoli fino alla concreta assegnazione dei compiti di ogni componente di *Jawa Druids*. In conformità allo standard ISO/IEC 12207, il processo di pianificazione è strutturato in due parti:

- ruoli di progetto;
- assegnazione dei ruoli.

### 4.2.2 Ruoli di progetto

I vari componenti del gruppo ricopriranno i seguenti ruoli:

- *Responsabile di Progetto*;
- *Amministratore di Progetto*;
- *Analista*;
- *Progettista*;
- *Programmatore*;
- *Verificatore*.

Il gruppo stabilirà un calendario in modo tale che ogni membro riesca a ricoprire almeno una volta ciascuno un ruolo in un periodo di tempo omogeneo senza gravare sullo svolgimento delle attività$_G$. Come previsto dal *Piano di Progetto 2.0.0* l'assegnazione di un ruolo comporta lo svolgimento di determinati compiti, inoltre il gruppo si impegnerà per eliminare eventuali conflitti: un componente non potrà mai redigere e successivamente verificare ciò che ha prodotto.

#### 4.2.2.1 Responsabile di Progetto

Il *Responsabile di Progetto*, ruolo fondamentale e presente per l'intera durata del lavoro, rappresenta il gruppo presso il proponente$_G$ ed i committenti$_G$. Il suo principale compito$_G$ è quello di coordinare la struttura ed organizzare il lavoro. In particolare questo ruolo comporta:

- il coordinamento dei membri del gruppo e dei compiti che devono portare a termine;
- la gestione della pianificazione, ossia l'attività$_G$ da svolgere e le relative scadenze da rispettare;
- avere la responsabilità della stima dei costi e dell'analisi dei rischi;
- la gestione delle relazioni che il gruppo intrattiene con i soggetti esterni;
- l'amministrazione delle risorse umane e dell'assegnazione dei ruoli;
- l'approvazione dei documenti.

#### 4.2.2.2 Amministratore di Progetto

L'*Amministratore del Progetto* ha il compito$_G$ di gestire, controllare e curare gli strumenti che il gruppo utilizza per lo svolgimento del proprio lavoro; è colui che garantisce l'affidabilità e l'efficacia dei mezzi. Questa figura persegue l'idea che la buona gestione dell'ambiente del lavoro favorisca la produttività, per questo motivo deve:

- amministrare le infrastrutture e i servizi necessari ai processi di supporto;
- gestire il versionamento e la configurazione dei prodotti;
- controllare la documentazione per assicurarsi che venga corretta, verificata ed approvata;
- facilitare il reperimento della documentazione;
- risolvere eventuali problemi legati alla gestione dei processi;
- redigere e manutenere le norme e le procedure$_G$ che regolano il lavoro;
- individuare gli strumenti utili all'automazione dei processi.

#### 4.2.2.3 Analista

L'*Analista* è presente nelle fasi iniziali del progetto, soprattutto quando avviene la stesura dell'*Analisi dei Requisiti* e il suo compito$_G$ è quello di evidenziare i punti fondamentali del problema in questione per comprenderne le sue peculiarità. Quindi la sua figura è fondamentale per la buona riuscita del lavoro, in quanto senza la sua analisi potrebbero essere presenti errori o mancanze nell'individuazione dei requisti che possono compromettere la successiva attività$_G$ di progettazione.
L'*Analista*:

- studia e definisce il problema in oggetto;
- analizza le richieste;
- fissa quali sono i requisiti$_G$ studiando i bisogni impliciti ed espliciti;
- analizza gli utenti e i casi d'uso;
- redige lo *Studio di Fattibilità* e l'*Analisi dei Requisiti*.

#### 4.2.2.4 Progettista

Il *Progettista* ha il compiti di sviluppare una soluzione che soddisfi i bisogni individuati dal lavoro dell'*Analista*. Lo scopo di questo compito$_G$, di natura sintetica, è quello di produrre un'architettura che modelli il problema a partire da un insieme di requisiti$_G$. Egli deve:

- applicare i principi noti e collaudati per produrre un'architettura coerente e consistente;
- produrre una soluzione sostenibile e realizzabile che rientri nei costi stabiliti dal preventivo;
- costruire una struttura che soddisfi i requisiti$_G$ e che sia aperta alla comprensione;
- limitare il più possibile il grado di accoppiamento tra le varie componenti;
- sforzarsi di cercare l'efficienza, la flessibilità e la riusabilità;
- elaborare una soluzione capace di interagire in modi diversi con l'ambiente in cui si pone e che sia sicura rispetto ad eventuali anomalie e intrusioni esterne;
- ricercare la massima disponibilità e affidabilità per l'architettura proposta.

#### 4.2.2.5 Programmatore

Il *Programmatore* è incaricato della codifica: il suo compito$_G$ è quello di implementare l'architettura prodotta dal *Progettista* in modo tale che aderisca alle specifiche. Egli è responsabile della manutenzione del codice creato, infatti i suoi compiti sono:

- codificare secondo le specifiche dettate dal *Progettista*, inoltre il codice prodotto deve essere documentato, versionabile e strutturato così da agevolare la futura manutenzione;
- creare le componenti che servono per la verifica e la validazione del codice;
- scrivere il *Manuale Utente* relativo al prodotto.

#### 4.2.2.6 Verificatore

Il *Verificatore* deve essere presente per tutta la durata del progetto e si occupa di controllare che le attività$_G$ svolte rispettino le norme e le attese prefissate. Egli deve:

- accertarsi che l'esecuzione delle attività$_G$ di processo non provochi errori;
- redigere la parte retrospettiva del *Piano di Qualifica* che chiarisce le verifiche e le prove effettuate.

### 4.2.3 Assegnazione dei compiti

La progressione del progetto può essere vista come il completamento sequenziale o parallelo di una serie di compiti, ognuno con scadenza temporale, i quali producono risultati utili per la realizzazione degli obiettivi. I compiti possono essere determinati da:

- contingenza;
- processi in atto;
- un insieme dei fattori precedenti.

La figura che si occupa della suddivisione ed assegnazione dei compiti è il *Responsabile di Progetto*, il quale:

- individua il compito$_G$ da svolgere;
- se ritiene il compito$_G$ complesso lo suddivide in diversi sotto-compiti;
- individua uno o più componenti del gruppo a cui assegnare il compito$_G$;
- crea le schede su Trello$_G$ e aggiorna la Timeline di GitKraken$_G$.

Di conseguenza i membri del gruppo devono impegnarsi per svolgere il compito$_G$ entro la data prefissata, avvisando nel caso in cui riscontrino problemi a rispettare le scadenze.

### 4.2.4 Trello e Gitkraken

Dopo aver suddiviso i compiti, il *Responsabile di Progetto* modificherà la pagina di Trello$_G$ del gruppo. Ogni scheda avrà la descrizione del compito$_G$ da svolgere, i componenti del gruppo che devono svolgerlo e la data di scadenza. Una volta che il task$_G$ è concluso viene spostata la sua scheda relativa nella sezione "Need revision" ovvero "Bisogno di revisione". Svolta l'attività$_G$ di verifica la scheda viene spostata in "Need approval", ossia che ha bisogno di approvazione, e infine in "Completed", ovvero completato.
Durante lo sviluppo i membri del gruppo possono aggiungere commenti ed informazioni utili allo svolgimento del compito$_G$. Se quest'ultimo è suddiviso in più parti sarà presente un elenco puntato che indica ogni suddivisione che l'addetto allo svolgimento potrà spuntare quando conclusa.
Per aiutare il gruppo viene fornita anche una rappresentazione grafica delle varie scadenze da rispettare grazie allo strumento Timeline di GitKraken$_G$.

### 4.2.5 Metriche

#### 4.2.5.1 MQPS01 Budget at Completion

Budget totale allocato per il progetto secondo il preventivo riportato nel *Piano di Progetto 2.0.0*. Si possono verificare tre casi:

- il budget preventivato è minore di quello effettivo al momento della valutazione, questo ci può indicare due cose:
  - il preventivo è stato sottostimato e quindi va rivisto;
  - il metodo di lavoro è sbagliato e c'è uno spreco di risorse;
- il budget preventivato e quello effettivo corrispondono, presumibilmente si sta lavorando nella maniera corretta;
- il budget preventivato è maggiore di quello effettivo, questo nella maggior parte dei casi indica che il preventivo è stato fatto con un margine troppo ampio.

#### 4.2.5.2 MQPS02 Planned Value

Metrica di utilità per il calcolo della SV (spiegata successivamente). Si tratta del valore del lavoro pianificato al momento del calcolo: corrisponde al denaro che si dovrebbe aver guadagnato in quel momento. La formula adottata è:

$$\text{PV} = \text{BAC} * \textit{\% lavoro pianificato}$$

- **PV** sigla di *Planned Value*;
- **BAC** sigla di *Budget at Completion*, riportato sopra;
- *% lavoro pianificato* al momento del calcolo.

#### 4.2.5.3 MQPS03 Actual Cost

Il denaro speso fino al momento del calcolo per lo svolgimento del progetto. E' molto importante misurare periodicamente questo valore in modo da riuscire a rendersi conto dell'andamento dei costi durante lo svolgimento del progetto e quindi di non superare la soglia del budget totale prima del completamento dello stesso. Per il calcolo si devono sommare tutti i costi già sostenuti.

#### 4.2.5.4 MQPS04 Earned Value

Metrica di utilità per il calcolo di SV e CV (spiegate successivamente). Si tratta del valore del lavoro fatto fino al momento del calcolo; corrisponde al denaro guadagnato fino a quel momento.

#### 4.2.5.5 MQPS05 Schedule Variance

Indica lo stato di avanzamento nello svolgimento del progetto rispetto a quanto pianificato. La formula adottata è:

$$SV = EV - PV$$

- **SV** è la sigla di *Schedule Variance*;
- **EV** è la sigla di *Earned Value*, cioè il valore effettivo di quanto prodotto alla data corrente;
- **PV** è la sigla di *Planned Value*, cioè il valore programmato per la data corrente.

#### 4.2.5.6 MQPS06 Cost Variance

Indica la differenza tra il costo di lavoro effettivamente completato e quindi "ribaltabile" al cliente come valore, ed il costo attualmente sostenuto. La formula adottata è:

$$CV = EV - AC$$

- **CV** sigla di *Cost Variance*;
- **EV** sigla di *Earned Value*, riportato sopra;
- **AC** sigla di *Actual Cost*, riportato sopra.

### 4.2.6 Strumenti

Per il supporto alla pianificazione del progetto e alla realizzazione dei diagrammi di Gannt$_G$ il gruppo ha deciso di utilizzare Fogli Google$_a$, software multipiattaforma con le seguenti funzionalità:

- la possibilità di modificare in maniera collaborativa e contemporanea i fogli di lavoro;
- la possibilità di creare diagrammi da tabelle.

## 4.3 Processo di miglioramento

### 4.3.1 Scopo

Il processo di miglioramento ha come scopo quello di assicurare il miglioramento del processo di sviluppo del prodotto, attraverso misurazioni e monitoraggi.

### 4.3.2 Descrizione

Il processo di miglioramento è impiegato per la manutenzione migliorativa dei processi durande lo sviluppo del prodotto software. Ciò si effetua tramite attività di valutazione e miglioramento.

### 4.3.3 Aspettative

Tramite tale sezione il gruppo si impegna a comprendere l'importanza del miglioramento continuo dei processi e dell'applicazione di tale processo.

### 4.3.4 Attività

#### 4.3.4.1 Implementazione del processo

Durante lo sviluppo del prodotto software, il gruppo deve adottare un procedimento di valutazione di processo, da applicare ad ogni processo, in modo tale che ciascuno di questi sia sviluppato e monitorato costantemente. E' necessario tenere traccia, attraverso la documentazione, di tale attività di valutazione, apportando delle modifche ai processi se ritenuto necessario. Per poter implementare in modo corretto e continuativo ciò, per ogni periodo, verrà riportata, nel *Piano di Qualifica*, in forma tabellare, una valutazione per ciasun processo applicato in tale periodo (??). In ciascuna tabella sarà presente:

- i problemi riscontrati;
- la descrizione di ciascun probelma;
- il livello di gravità attribuito ad ogni problema, attraverso un'indice numerico che varia da 1 a 3;
- la soluzione individuata dal gruppo per sanare il problema.

## 4.4 Formazione

I membri del gruppo devono provvedere alla propria formazione autonoma studiando le tecnologie usate e colmando eventuali carenze, per poter garantire una qualità di lavoro che rispetti le aspettative. Il gruppo fa riferimento alla seguente documentazione e a quella elencata nei riferimenti dei documenti redatti:

- *LaTeX:* https://www.latex-project.org/ e https://www.overleaf.com/learn/latex/Main_Page;
- *TeXStudio:* https://www.texstudio.org/;

- *Vue.js*$_G$: https://vuejs.org/;
- *Spring*$_G$: https://spring.io/;
- *Java*$_G$: https://docs.oracle.com/en/java/;
- *Leaftletjs*$_G$: https://leafletjs.com/reference-1.7.1.htm;
- *Mqtt*$_G$: https://mqtt.org/;
- *Scikit-learn*$_G$: https://scikit-learn.org/stable/index.html;
- *Python*$_G$: https://www.python.org/doc/;
- *Kafka*$_G$: https://kafka.apache.org/documentation/;
- *Github*$_G$: https://help.github.com/;
- *Git*$_G$: https://git-scm.com/;
- *Git-flow*$_G$: https://github.com/nvie/gitflow;
- *Farfalla Project*$_G$: https://farfalla-project.org/readability_static/.

# 5 Standard ISO/IEC 9126

ISO/IEC 9126$_G$ è uno standard internazionale per la valutazione della qualità del software. Tale standard definisce i seguenti modelli:

- **Qualità interna:** definisce le metriche applicabili al codice sorgente del software non eseguibile durante le fasi di progettazione e codifica. Tramite tale metriche è possibile identificare eventuali problemi che potrebbero minacciare la qualità del software prima che esso vada in esecuzione. Inoltre possono misurare il comportamento del prodotto finale tramite simulazione. Il rilevamento della qualità interna avviene tramite l'analisi statica. Attraverso le metriche interne è possibile, inoltre, prevedere il livello di qualità esterna ed in uso del prodotto software finale, in quanto, idealmente, la qualità interna determina la qualità esterna. Infine, le metriche interne si applicano anche alla documentazione del prodotto;
- **Qualità esterna:** definisce le metriche applicabili al software in esecuzione che ne misurano i comportamenti attraverso i test, l'operatività, l'osservazione durante l'esecuzione, in funzione degli obiettivi stabiliti. Il rilevamento della qualità esterna avviene tramite l'analisi dinamica. Idealmente, la qualità esterna determina la qualità in uso;
- **Qualità in uso:** definisce le metriche applicabili solo quando il prodotto software è finito e usato in condizioni reali. Inoltre misurano il grado con cui il prodotto permette agli utenti di svolgere le proprie attività$_G$ con efficacia, sicurezza, produttività e soddisfazione nel contesto operativo previsto.

## 5.1 Modello della qualità esterna ed interna del software

Il modello della qualità del software descritto dallo standard ISO/IEC 9126$_G$ definisce le caratteristiche e le sotto-caratteristiche del software, ciascuna misurabile da metriche interne o esterne. le caratteristiche sono quelle riportate qui di seguito.

1. **Funzionalità:** capacità del software di di fornire le funzioni appropriate, necessarie per soddisfare i bisogni espressi nell'*Analisi dei Requisiti 3.0.0* e per operare in un determinato contesto. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:
    - **Adeguatezza:** capacità di fornire un appropriato insieme di funzioni che permettano di raggiungere gli obiettivi prefissati;
    - **Accuratezza:** capacità di fornire i risultati o gli effetti attesi in maniera corretta e con la precisione richiesta;
    - **Interoperabilità:** capacità di interagire con uno o più sistemi specificati;

- **Sicurezza:** capacità di proteggere le informazioni ed i dati;
- **Aderenza:** capacità di aderire agli standard.

2. **Affidabilità:** capacità di un prodotto software di mantenere il livello di prestazione quando viene usato in condizioni specifiche. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:
   - **Maturità:** capacità di evitare che si verifichino errori o risultati non corretti in fase di esecuzione;
   - **Tolleranza ai guasti:** capacità di mantenere il livello di prestazione in caso di errori nel software o usi scorretti del prodotto finale;
   - **Recuperabilità:** capacità di ripristinare il livello di prestazioni e di recuperare i dati rilevanti in caso di errori o malfunzionamenti;
   - **Aderenza:** capacità di aderire agli standard.
3. **Usabilità:** capacità di un prodotto software di essere comprensibile ed attraente per l'utente, sotto determinate condizioni. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:
   - **Comprensibilità:** capacità di permettere all'utente di capire la sua funzionalità e come poterla utilizzare con successo;
   - **Apprendibilità:** capacità di permettere all'utente di imparare ad usare l'applicazione;
   - **Operabilità:** capacità di permettere all'utente di utilizzarlo e di controllarlo;
   - **Attrattività:** capacità di risultare attraente per l'utente;
   - **Aderenza:** capacità di aderire agli standard.
4. **Efficienza:** capacità di un prodotto software di realizzare le funzioni richieste nel minor tempo possibile e sfruttando al meglio le risorse necessarie, quando opera in determinate condizioni. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:
   - **Comportamento rispetto al tempo:** capacità di fornire appropriati tempi di risposta, tempi di elaborazione e quantità di lavoro eseguendo le funzionalità previste sotto determinate condizioni di utilizzo;
   - **Utilizzo delle risorse:** capacità di utilizzare un appropriato numero e tipo di risorse in maniera adeguata;
   - **Aderenza:** capacità di aderire agli standard.
5. **Manutenibilità:** capacità di un prodotto software di essere modificato. Le modifiche possono includere correzioni, adattamenti o miglioramenti del software. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:

- **Analizzabilità:** capacità di poter effettuare la diagnosi sul software ed individuare le cause di errori o malfunzionamenti;
- **Modificabilità:** capacità di consentire lo sviluppo di modifiche al software originale. L'implementazione include modifiche al codice, alla progettazione ed alla documentazione;
- **Stabilità:** capacità di evitare effetti non desiderati a seguito di modifiche al software;
- **Testabilità:** capacità di consentire la verifica e validazione del software modificato, cioè di eseguire i test;
- **Aderenza:** capacità di aderire agli standard.

6. **Portabilità:** capacità di un prodotto software di poter essere trasportato da un ambiente hardware/software ad un altro. Più specificatamente, il software deve soddisfare le seguenti caratteristiche:

   - **Adattabilità:** capacità di adattarsi a diversi ambienti senza richiedere azioni specifiche diverse da quelle previste dal software per tali attività$_G$;
   - **Installabilità:** capacità di essere installato in un determinato ambiente;
   - **Coesistenza:** capacità di coesistere con altre applicazioni indipendenti in ambienti comuni e di condividere le risorse;
   - **Sostituibilità:** capacità di sostituire un altro software specifico indipendente per lo stesso scopo e nello stesso ambiente;
   - **Aderenza:** capacità di aderire agli standard.

## 5.2 Modello della qualità in uso del software

Questo modello consente agli utenti di ottenere risultati specifici con:

- **Efficacia:** capacità di permettere all'utente di raggiungere determinati obiettivi con accuratezza e completezza in uno specifico contesto di utilizzo;
- **Produttività:** capacità di permettere all'utente di impiegare un numero definito di risorse, in relazione all'efficienza raggiunta in uno specifico contesto di utilizzo;
- **Sicurezza:** capacità di raggiungere un livello accettabile di rischio rispetto a danni nei confronti di persone, apparecchiature tecniche, software ed ambiente d'uso;
- **Soddisfazione:** capacità di soddisfare gli utenti.

# 6 Standard ISO/IEC 15504

Il modello ISO/IEC 15504$_G$, noto anche come SPICE, acronimo di *Software Process Improvement and Capability Determination* ("capability" è una parola chiave, ovvero la *"capacità"* intesa come abilità di un processo nel raggiungere un obiettivo). SPICE è uno standard di riferimento per valutare la qualità dei processi con il fine di migliorarli. Questo standard permette di misurare indipendentemente la *capacità* di ogni processo tramite degli attributi, studiando i risultati che si ottengono eseguendolo, tali risultati devono essere ripetibili, oggettivi e comparabili perchè possano contribuire, appunto, al miglioramento dei processi. Gli attributi associati alle capacità di ogni processo sono:

- **Process definition:** indica in quale modo il processo fa riferimento agli standard;
- **Process performance:** indica il risultato rispetto al raggiungimento degli obiettivi fissati;
- **Process distribution:** indica quanto si discosta il processo standard rispetto ad un processo definito in grado di raggiungere sempre gli stessi risultati;
- **Process measurement:** indica il grado in cui i risultati delle misurazioni sono utilizzati per garantire che il processo raggiunga gli obiettivi fissati;
- **Process control:** indica quanto il processo è stabile, capace e predicibile entro certo limiti definiti;
- **Process change:** indica in quale misura le modifiche da apportare al processo sono identificate grazie ad una fase di analisi delle performance e allo studio di approcci innovativi;
- **Process improvement:** indica in che modo i cambiamenti alle performance e alla definizione del processo hanno un impatto effettivo, che porta a raggiungere importanti obiettivi di miglioramento al processo;
- **Performance management:** indica il grado di organizzazione con cui sono raggiunti gli obiettivi fissati;
- **Work product management:** indica in quale misura i prodotti sono gestiti correttamente per quanto riguarda documentazione, controllo e verifica.

Ogni attributo viene misurato e classificato con uno dei seguenti livelli:

- **N - not implemented:** il processo non implementa tale attributo o lo fa in modo molto carente;

- **P - partially implemented:** il processo implementa parzialmente tale attributo tramite un approccio sistematico, tuttavia alcuni aspetti non sono ancora prevedibili;
- **L - largely implemented:** il processo possiede significativamente tale attributo tramite un approccio sistematico, ma il modo in cui è attuato varia nelle diverse unità;
- **F - fully implemented:** l'attributo è stato completamente ottenuto grazie ad un approccio sistematico e l'attuazione è uguale in tutte le unità.

Perciò, in base alla classificazione degli attributi, ad un processo viene assegnato un livello di capacità pari a:

- **0 - Incomplete:** il processo è incompleto in quanto non è stato implementato o fallisce nel raggiungere il proprio obiettivo. Questo livello non ha alcun attributo associato;
- **1 - Performed:** il processo è stato implementato e ha successo nel raggiungere il proprio obiettivo. L'attributo associato a questo livello è "process performance";
- **2 - Managed:** il processo, che già apparteneva al livello *performed*, è implementato in maniera organizzata tramite pianificazione, controllo e correzioni; i suoi prodotti sono sicuri. Gli attributi associati a questo livello sono "performance management" e "work product management";
- **3 - Established:** il processo, che già apparteneva al livello *managed*, è stato implementato come processo definito in grado di raggiungere sempre gli stessi risultati. Gli attributi associati a questo livello sono "process definition" e "process distribution";
- **4 - Predictable:** il processo, che già apparteneva al livello *established*, opera entro limiti definiti per raggiungere i propri risultati. Gli attributi associati a questo livello sono "process control" e "process measurement";
- **5 - Optimizing:** il processo, che già apparteneva al livello *predictable*, è oggetto di miglioramento continuo per raggiungere gli obiettivi di progetto/aziendali. Gli attributi associati a questo livello sono "process change" e "process improvement".

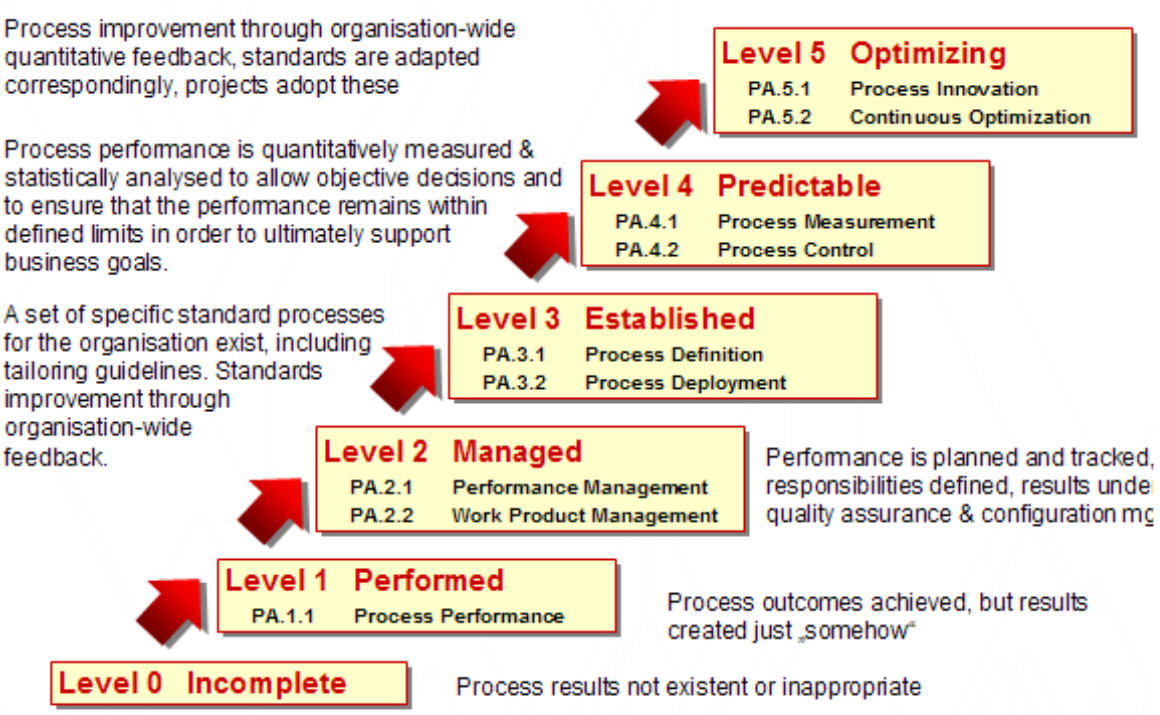


Figura 6.1: SPICE Capability (Immagine tratta da HM&S)

# 7 Formazione

## 7.1 Processo di formazione

Tramite il processo di formazione si intendono fornire materiali e strumenti idonei a rendere il gruppo di lavoro qualificato allo sviluppo del prodotto software. Questo processo consiste nelle seguenti attività$_G$:

1. **Piano di formazione**;
2. **Ricerca del materiale**;
3. **Inizio della formazione**.

### 7.1.1 Piano di formazione

In questa prima fase il gruppo di lavoro, assieme al *Responsabile di progetto*, discuteranno e decideranno su quali siano le abilità principali e necessarie per diventare sviluppatori qualificati nell'ambito del prodotto software. Questa decisione sarà basata sulle richieste proposte nel capitolato d'appalto$_G$ del committente$_G$.

### 7.1.2 Ricerca del materiale

Una volta definito il piano da seguire si prosegue con la ricerca attiva del materiale da parte del gruppo. Questo lavoro può essere svolto mediante tre macro-categorie:

- **Libri**: quindi la ricerca tramite libri testuali, o digitali, contenenti nozioni sugli argomenti da imparare;
- **Azienda**: ovvero la richiesta di materiale specifico all'azienda proponente$_G$ del software da sviluppare;
- **Internet**: quindi cercando su forum o siti specializzati informazioni pertinenti tutto ciò che riguarda il progetto.

### 7.1.3 Inizio della formazione

Raccolto il materiale adatto, il gruppo inizierà la propria formazione sia personale, sia collettiva nel caso qualche componente avesse problemi con alcuni concetti.