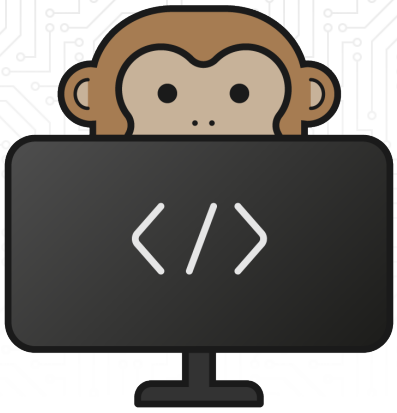

ProApes

proapes11@gmail.com

Norme di Progetto

Versione	4.0.0-1.10
Data approvazione	2020-05-14
Responsabile	Federico Carboni
Redattori	Igor Biolcati Rinaldi Alessandro Discalzi Valentina Signor
Verificatori	Fiammetta Cannavò Alessandro Discalzi Valentina Signor
Stato	Approvato
Lista distribuzione	*ProApes* *Prof. Tullio Vardanega* *Prof. Riccardo Cardin*
Uso	Interno

Sommario

Il presente documento contiene le regole, gli strumenti e le convenzioni a cui il gruppo *ProApes* ha stabilito di attenersi durante l'intero processo di sviluppo del progetto *Predire in Grafana*.

Diario delle Modifiche

Versione	Data	Modifica	Autore	Ruolo
v4.0.0-1.10	2020-05-14	*Approvazione del documento per RA*	Federico Carboni	*Responsabile di Progetto*
v3.1.0-1.10	2020-05-12	*Revisione complessiva finale di coerenza e coesione (*Verificatore*: Fiammetta Cannavò)*	Valentina Signor	*Amministratore di Progetto*
v3.0.3-1.9	2020-05-10	*Aggiornata §2.2.7 (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v3.0.2-1.8	2020-05-03	*Aggiunta sigla per Gestione dei cambiamenti in §3.6.5 (*Verificatore*: Alessandro Discalzi)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v3.0.1-1.8	2020-05-01	*Aggiornamento registro delle modifiche e aggiunte sigle per i manuali in §3.1.8.4 (*Verificatore*: Fiammetta Cannavò)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v3.0.0-1.5	2020-04-13	*Approvazione del documento per RQ*	Giacomo Piran	*Responsabile di Progetto*
v2.1.1-0.5	2020-04-11	*Riviste §2.2.6.5, §3.2.4.1 e §3.2.4.2 (*Verificatore*: Fiammetta Cannavò)*	Valentina Signor	*Amministratore di Progetto*
v2.1.0-0.4	2020-04-09	*Revisione complessiva finale di coerenza e coesione (*Verificatore*: Igor Biolcati Rinaldi)*	Fiammetta Cannavò	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v2.0.7-0.4	2020-04-08	*Dettagliati competitors e aggiunto nuovo canale Slack in §4.1.4.4 (*Verificatore*: Igor Biolcati Rinaldi)*	Fiammetta Cannavò	*Amministratore di Progetto*
v2.0.6-0.4	2020-04-07	*Snellita struttura indice e sistemati riferimenti a sezioni di altri documenti (*Verificatore*: Igor Biolcati Rinaldi)*	Fiammetta Cannavò	*Amministratore di Progetto*
v2.0.5-0.1	2020-03-23	*Aggiunta sezione §4.1.4.6 (*Verificatore*: Giacomo Piran)*	Valentina Signor	*Amministratore di Progetto*
v2.0.4-0.1	2020-04-22	*Aggiunta §3.4.10 (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v2.0.3-0.1	2020-04-20	*Aggiunti diagrammi delle attività in §3.4 e §3.5 (*Verificatore*: Federico Carboni)*	Valentina Signor	*Amministratore di Progetto*
v2.0.2-0.1	2020-03-18	*Ampliata §2.2.5 (*Verificatore*: Giacomo Piran)*	Valentina Signor	*Amministratore di Progetto*
v2.0.1-0.1	2020-03-17	*Aggiunte metriche di qualità di prodotto software a §D.2 (*Verificatore*: Giacomo Piran)*	Valentina Signor	*Amministratore di Progetto*
v2.0.0	2020-03-09	*Approvazione del documento per RP*	Francesco Bari	*Responsabile di Progetto*
v1.7.0	2020-03-08	*Revisione complessiva finale di coerenza e coesione (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v1.6.3	2020-03-08	*Ampliate §3.2.5, §3.2.6: aggiunta repo globale (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*
v1.6.2	2020-03-06	*Ampliate §2.2.6.8, §2.2.6.9, §2.2.6.10 3.3.4.2, §3.2.5.2 (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*
v1.6.1	2020-03-01	*Aggiunta §4.1.5.7 (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.6.0	2020-02-27	*Revisione complessiva di coerenza e coesione (*Verificatore*: Valentina Signor)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.5.4	2020-02-26	*Riviste §A, §B, §C (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*
v1.5.3	2020-02-25	*Aggiunta §4.3 (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.5.2	2020-02-25	*Aggiunte §4.2, §4.4 (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.5.1	2020-02-24	*Corretta §3.3.7, aggiunta §3.3.6 (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.5.0	2020-02-23	*Revisione complessiva di coerenza e coesione (*Verificatore*: Giacomo Piran)*	Federico Carboni	*Amministratore di Progetto*
v1.4.3	2020-02-22	*Aggiunte §4.1.5.2, §4.1.5.3 (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v1.4.2	2020-02-22	*Aggiunte §4.1, § 4.1.4.2, §4.1.4.3 (*Verificatore*: Giacomo Piran)*	Federico Carboni	*Amministratore di Progetto*
v1.4.1	2020-02-21	*Modificate §4.1.5.5, §4.1.5.6 (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.4.0	2020-02-21	*Revisione complessiva di coerenza e coesione (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.3.3	2020-02-20	*Terminata §3.2.4.1, completata §3.6 (*Verificatore*: Valentina Signor)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.3.2	2020-02-18	*Impostata §3.2.4.1 e iniziata la sua compilazione (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*
v1.3.1	2020-02-15	*Corretta §3.2.4.1 (*Verificatore*: Valentina Signor)*	Federico Carboni	*Amministratore di Progetto*
v1.3.0	2020-02-14	*Revisione complessiva di coerenza e coesione (*Verificatore*: Valentina Signor)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.2.2	2020-02-13	*Estese §3.1.8, §3.4, §3.4.5, parzialmente §3.3.4 (*Verificatore*: Fiammetta Cannavò)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.2.1	2020-02-11	*Correzione di §1, estensione sezione §2.2.6.7 (*Verificatore*: Fiammetta Cannavò)*	Federico Carboni	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v1.2.0	2020-02-11	*Revisione complessiva di coerenza e coesione (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.1.3	2020-02-10	*Conclusa §2.2.5 (*Verificatore*: Giacomo Piran)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.1.2	2020-02-08	*Aggiunte §2.2.5.5 e §2.2.5.9 (*Verificatore*: Giacomo Piran)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.1.1	2020-02-08	*Estese §2.1, §2.2.5.7 (*Verificatore*: Giacomo Piran)*	Federico Carboni	*Amministratore di Progetto*
v1.1.0	2020-02-07	*Revisione complessiva di coerenza e coesione (*Verificatore*: Fiammetta Cannavò)*	Alessandro Discalzi	*Amministratore di Progetto*
v1.03	2020-02-06	*Estese §3.1.10, §3.1.8, §3.4, §3.5 (*Verificatore*: Fiammetta Cannavò)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.0.2	2020-02-05	*Aggiunte §2.1.4, §2.2.4.7, §3.1.10 (*Verificatore*: Fiammetta Cannavò)*	Federico Carboni	*Amministratore di Progetto*
v1.0.1	2020-02-04	*Effettuate correzioni in §2, §3 (*Verificatore*: Fiammetta Cannavò)*	Igor Biolcati Rinaldi	*Amministratore di Progetto*
v1.0.0	2019-01-14	*Approvazione del documento per RR*	Igor Biolcati Rinaldi	*Responsabile di Progetto*
v0.7.0	2019-01-13	*Revisione finale complessiva di coerenza e coesione (*Verificatore*: Fiammetta Cannavò)*	Francesco Bari	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v0.6.1	2019-01-12	*Aggiunte §A, §B, §C dal Piano di Qualifica 1.0.0 (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.6.0	2019-12-23	*Revisione di quanto aggiunto in §2 e §3 (*Verificatore*: Francesco Bari)*	Federico Carboni	*Verificatore*
v0.5.0	2019-12-23	*Revisione complessiva di coerenza e coesione (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.4.3	2019-12-22	*Aggiunte §3.4, §3.5 (*Verificatore*: Fiammetta Cannavò)*	Giacomo Piran	*Amministratore di Progetto*
v0.4.2	2019-12-20	*Aggiunta §3.3 (*Verificatore*: Federico Carboni)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.4.1	2019-12-19	*Aggiunte §2.2.6.7, §2.2.7 (*Verificatore*: Fiammetta Cannavò)*	Valentina Signor	*Amministratore di Progetto*
v0.4.0	2019-12-18	*Revisione complessiva di coerenza e coesione (*Verificatore*: Fiammetta Cannavò)*	Alessandro Discalzi	*Amministratore di Progetto*
v0.3.3	2019-12-17	*Aggiunta §3.2 (*Verificatore*: Francesco Bari)*	Giacomo Piran	*Amministratore di Progetto*
v0.3.2	2019-12-16	*Aggiunte §4.1.5.6, §4.2.5, §4.4 (*Verificatore*: Francesco Bari)*	Giacomo Piran	*Amministratore di Progetto*
v0.3.1	2019-12-15	*Aggiunte §2.2.5, §2.2.6 (*Verificatore*: Francesco Bari)*	Valentina Signor	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v0.3.0	2019-12-13	*Revisione complessiva di coerenza e coesione (*Verificatore*: Francesco Bari)*	Francesco Bari	*Amministratore di Progetto*
v0.2.4	2019-12-13	*Aggiunte §3.1.8, §3.1.8.5, §3.1.10 (*Verificatore*: Federico Carboni)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.2.3	2019-12-13	*Aggiunta §2.2.4 (*Verificatore*: Federico Carboni)*	Valentina Signor	*Amministratore di Progetto*
v0.2.2	2019-12-12	*Aggiunte §4.1.5.4, §4.1.5.5 (*Verificatore*: Federico Carboni)*	Giacomo Piran	*Amministratore di Progetto*
v0.2.1	2019-12-11	*Aggiunte le sezioni §4.1.4.5, §4.2.4, §4.1.5.1 (*Verificatore*: Federico Carboni)*	Giacomo Piran	*Amministratore di Progetto*
v0.2.0	2019-12-11	*Revisione complessiva di coerenza e coesione (Verificatore*: Federico Carboni)	Valentina Signor	*Verificatore*
v0.1.4	2019-12-11	*Aggiunte §3.1.5, §3.1.6, §3.1.7.1, §3.1.7.2 (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.1.3	2019-12-09	*Aggiunte §3.1.2, §3.1.3, §3.1.4 (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.1.2	2019-12-06	*Aggiunte §2.1.5, §2.1.9, §2.2.1, §2.2.2, §2.2.3 (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.1.1	2019-12-06	*Aggiunta §4.1.4.4 (*Verificatore*: Francesco Bari)*	Giacomo Piran	*Amministratore di Progetto*

Versione	Data	Modifica	Autore	Ruolo
v0.1.0	2019-12-05	*Revisione complessiva di coerenza e coesione (*Verificatore*: Francesco Bari)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.0.4	2019-12-05	*Aggiunte §2.1.1, §2.1.2, §2.1.3, §2.1.6 (*Verificatore*: Francesco Bari)*	Valentina Signor	*Amministratore di Progetto*
v0.0.3	2019-12-03	*Inizio stesura §2 (*Verificatore*: Francesco Bari)*	Valentina Signor	*Amministratore di Progetto*
v0.0.2	2019-12-03	*Inizio stesura §3 (*Verificatore*: Federico Carboni)*	Fiammetta Cannavò	*Amministratore di Progetto*
v0.0.1	2019-12-02	*Inizio stesura dello scheletro del documento e di §4 (*Verificatore*: Federico Carboni)*	Giacomo Piran	*Amministratore di Progetto*

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento stabilisce tutte le regole e le procedure fondamentali che ciascun membro di *ProApes* deve rispettare per ottenere un'organizzazione uniforme ed efficiente dei file prodotti.
Durante l'intero svolgimento del progetto tutti i membri del gruppo sono obbligati a visionare ed a rispettare le norme che vi sono descritte.
Il documento allo stato attuale risulta essere incompleto in quanto redatto seguendo una filosofia di tipo incrementale, volta a normare passo passo ogni decisione discussa e concordata preventivamente. In questo modo vengono decise prima le norme ritenute più urgenti, e solo poi quelle che saranno necessarie per le attività da svolgersi in seguito.
Le norme già presenti potranno essere sottoposte a cambiamenti (aggiunte, rimozioni, modifiche), che dovranno essere comunicate [1] a ciascun componente del gruppo dal *Responsabile di Progetto*.

1.2 Scopo del prodotto

Il capitolato$_{\mathbf{G}}$ C4 - *Predire in Grafana* nasce dall'esigenza, a seguito dell'applicazione di una politica di tipo DevOps$_{\mathbf{G}}$ durante il ciclo di vita$_{\mathbf{G}}$ del software, di effettuare un monitoraggio costante delle applicazioni e delle informazioni ivi contenute. A tal fine il gruppo *ProApes* si propone di sviluppare per l'azienda *Zucchetti S.p.A.* un plug-in$_{\mathbf{G}}$ da affiancare allo strumento di monitoraggio *Grafana*$_{\boldsymbol{G}}$ che applichi le tecniche di *SVM*$_{\boldsymbol{G}}$ e *Regressione Lineare*$_{\boldsymbol{G}}$ sul flusso dei dati ricevuti per allarmi o segnalazioni tra gli operatori del servizio Cloud e la linea di produzione del software.

1.3 Glossario

All'interno del documento sono presenti termini che possono presentare significati ambigui o incongruenti a seconda del contesto. Al fine quindi di evitare l'insorgere d'incomprensioni viene fornito un glossario individuabile nel file *Glossario 4.0.0-1.10* contenente i suddetti termini e la loro spiegazione.
Nella seguente documentazione per favorire maggiore chiarezza ed evitare inutili ridondanze tali parole vengono indicate mettendo una "G" a pedice di ogni prima occorrenza del termine che si incontri ad ogni inizio di sezione.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato d'Appalto C4**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C4.pdf;

1.4.2 Riferimenti informativi

- **Seminario per approfondimenti tecnici sul capitolato C4**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/C4a.pdf;
- **Software Engineering - Ian Sommerville - 9 th Edition (2010)**:
  - Capitolo 25 - Configuration management.

[1] Le norme riguardanti le comunicazioni interne sono definite nell'apposita sotto-sezione in §4.1.4.4

- **Libro ProGit**:
  https://git-scm.com/book/it/v2
  - Capitolo 1 - Per Iniziare;
  - Capitolo 2 - Git Basics;
  - Capitolo 3 - Git Branching;
  - Capitolo 4 - Git on the Server;
  - Capitolo 6 - *GitHub*$_{\mathbf{G}}$.
- **Materiale didattico del corso di Tecnologie Open-Source**:
  - Lezione 5, GIT;
  - Laboratorio 2, Git Work Flow e GitHub ITS;
  - Lezione 10, Continuous Integration$_{\mathbf{G}}$.
- **Processi SW - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/L03.pdf
  - Standard di processo, slide 10;
  - ISO/IEC 12207, Processi primari, di supporto e organizzativi, slide 12 - 16;
  - Alcune attività di processo, slide 20 - 22.
- **Amministrazione di progetto - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/FC01.pdf
  - Documentazione di progetto, slide 6 - 7;
  - Ambiente di lavoro, configurazione, gestione delle modifiche e controllo di versione, slide 8 - 24;
  - Norme di Progetto, slide 26 - 29.
- **L'Arte di scrivere con LaTeX - Lorenzo Pantieri & Tommaso Gordini**:
  http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf
  - Capitolo 3 - Basi;
  - Capitolo 4 - Testo;
  - Capitolo 5 - Matematica;
  - Capitolo 6 - Tabelle e figure.
- **Standard ISO/IEC 12207:1995**:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- **La qualità del software secondo il modello ISO/IEC9126 - Ercole F. Colonese - Versione 2.0**:
  http://www.colonese.it/00-Manuali_Pubblicatii/07-ISO-IEC9126_v2.pdf
  - Capitolo 2 - Il modello ISO/IEC 9126;
  - Capitolo 3 - Le metriche della qualità del software.
- **Standard ISO/IEC 9126**:
  - https://it.wikipedia.org/wiki/ISO/IEC_9126;
  - https://en.wikipedia.org/wiki/ISO/IEC_9126.
- **Standard ISO/IEC 15504**:
  - https://en.wikipedia.org/wiki/ISO/IEC_15504;

  - SWEBOK guide 3.0, p. B-16.

- **Metriche di progetto**:
  https://it.wikipedia.org/wiki/Metriche_di_progetto;
- **Ciclo di Deming**:
  https://it.wikipedia.org/wiki/Ciclo_di_Deming;
- **Immagine presente in §B**:
  https://isigmagroup.com/event/2697/.
- **Airbnb JavaScript Style Guide**:
  https://github.com/airbnb/javascript;
  - Capitolo 2 - References;
  - Capitolo 4 - Arrays;
  - Capitolo 8 - Arrow Functions;
  - Capitolo 10 - Modules.
- **Airbnb React/JSX Style Guide**:
  https://github.com/airbnb/javascript/tree/master/react;
- **Google HTML/CSS Style Guide**:
  https://google.github.io/styleguide/htmlcssguide.html
  - Capitolo 3 - HTML;
  - Capitolo 4 - CSS.
- **Diagrammi delle attività - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02b.pdf
  - Diagrammi delle attività, slide 6 - 10;
  - Azioni, slide 12 - 14;
  - Partizioni, slide 15;
  - Segnali, slide 16 - 17;
  - Nodi di terminazione, slide 21.
- **Diagrammi delle classi - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01b.pdf
  - Diagrammi delle classi, slide 6 - 9;
  - Proprietà, slide 11 - 14;
  - Operazioni, slide 16 - 17;
  - Relazione di dipendenza, slide 21 - 23;
  - Aggregazione e Composizione, slide 25 - 26;
  - Classi di Associazione, slide 27 - 28;
  - Generalizzazione, slide 29 - 30;
  - Classi astratte e Interfacce, slide 31 - 33.
- **Diagrammi dei package - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01c.pdf
  - Cos'è un Package, slide 6 - 8;
  - Package e Dipendenze, slide 13 - 15.

- **Diagrammi di sequenza - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02a.pdf
  - Diagrammi di sequenza, slide 6;
  - Partecipanti, slide 8;
  - Messaggi (Segnali), 9 - 16;
  - Cicli e Condizioni, 18 - 19.
- **Diagrammi dei casi d'uso - Materiale didattico del corso di Ingegneria del Software**:
  https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E03.pdf
  - Diagrammi dei casi d'uso, slide 17 - 18;
  - Use Case: Inclusione, slide 20 - 21;
  - Use Case: Estensione, slide 22 - 24;
  - Use Case: Generalizzazione, slide 26 - 27.

2 Processi Primari

2.1 Fornitura

2.1.1 Scopo

Il processo$_{\mathbf{G}}$ di Fornitura, come stabilito dallo standard$_{\mathbf{G}}$ ISO/IEC 12207:1995$_{\mathbf{G}}$, determina ogni compito$_{\mathbf{G}}$, attività$_{\mathbf{G}}$ e risorsa necessaria allo svolgimento del progetto. Una volta comprese le richieste del proponente$_{\mathbf{G}}$, redatto lo *Studio di Fattibilità*, e definito un accordo contrattuale col proponente stesso, il processo potrà allora essere avviato. Verranno stabilite le procedure e le risorse necessarie, che confluiranno nel *Piano di Progetto*, ponendo così le basi per la realizzazione e consegna del prodotto finale.
Il processo di Fornitura si compone delle seguenti fasi:

- avvio;
- approntamento di risposte alle richieste;
- contrattazione;
- pianificazione;
- esecuzione e controllo;
- revisione e valutazione;
- consegna e completamento.

2.1.2 Descrizione

La presente sezione contiene tutte le norme che ogni membro del gruppo è tenuto a seguire, durante le varie fasi di svolgimento del progetto, per poter divenire fornitori$_{\mathbf{G}}$ del proponente *Zucchetti S.p.A.* e dei committenti$_{\mathbf{G}}$ *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*.
Nelle sottosezioni che seguono verranno normate le attività riguardanti:

- la definizione di un accordo contrattuale, per gestire i rapporti con il proponente, la consegna e la manutenzione del prodotto finale;
- l'analisi e successiva redazione dello *Studio di Fattibilità*, individuando i rischi e le criticità inerenti alla richiesta d'appalto;
- l'identificazione dei rischi in cui il fornitore pensa di poter incorrere durante lo svolgimento del progetto.

2.1.3 Aspettative

Per ottenere un riscontro efficace sul lavoro svolto, il gruppo *ProApes* si propone di instaurare e mantenere un dialogo continuo[2] e un costante rapporto collaborativo con l'azienda *Zucchetti S.p.A.*, nello specifico nella persona del referente indicato *Dott. Gregorio Piccoli*.

2.1.4 Rapporti con l'azienda proponente *Zucchetti S.p.A.*

Tale rapporto permette di arrivare ad un contratto fra le parti per:

- determinare aspetti chiave per far fronte ai bisogni del proponente;
- individuare la strategia da adottare per soddisfare i bisogni e i vincoli del proponente;
- stilare requisiti e vincoli sui processi;

[2]Le modalità di comunicazione esterna sono indicate nell'apposita sotto-sezione in §4.1.4.4

- stimare le tempistiche di lavoro;
- incentivare una verifica continua;
- chiarire eventuali dubbi emersi;
- accordarsi sulla qualifica del prodotto.

Avvenuta la consegna del prodotto il gruppo, a meno di diversi accordi con il proponente, non seguirà l'attività di manutenzione dello stesso.

2.1.5 Materiale fornito

Di seguito vengono elencati i documenti consegnati all'azienda proponente ed ai committenti *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*. Essi assicurano la completa trasparenza delle attività di Analisi, Pianificazione, Verifica, Validazione e Controllo di qualità durante l'intero ciclo di vita$_{\mathbf{G}}$ del progetto. Nello specifico:

- ***Analisi dei Requisiti 4.0.0-1.10*:** contiene l'analisi dei casi d'uso e dei requisiti (diretti ed indiretti, espliciti e non); tale documento punta a:
  - evitare l'insorgere di ambiguità riguardanti il capitolato;
  - definire nel dettaglio le funzionalità che saranno offerte dal prodotto.
- ***Piano di Progetto 4.0.0-1.10*:** contiene la pianificazione preventiva dei tempi e delle attività, l'analisi dei rischi e il consuntivo di periodo, oltre alla data e ai costi previsti per la realizzazione del prodotto finale;
- ***Piano di Qualifica 4.0.0-1.10*:** assicura al proponente una qualità di processi e prodotti adeguata alle sue aspettative, fornendo le modalità da adottare in sede di verifica e validazione;
- ***Proof of Concept*$_{\mathbf{G}}$ e *Technology Baseline*$_{\mathbf{G}}$:** nell'ambito della Progettazione architetturale, definiscono una panoramica ad alto livello dell'applicazione che il gruppo *ProApes* realizzerà e delle tecnologie impiegate (vedi §2.2.5.5);
- ***Product Baseline*$_{\mathbf{G}}$:** nell'ambito dell'attività di Progettazione di dettaglio, definisce l'insieme di classi, metodi, attributi e scelte implementative a livello tecnico (vedi §2.2.5.5).

Alla documentazione è allegata una **Lettera di Presentazione** con cui i membri del gruppo *ProApes* formalizzano il loro impegno nel portare a termine il capitolato prescelto rispettandone i requisiti minimi e consegnando il prodotto finito entro i termini definiti dalla lettera stessa.
Il prodotto software finale idoneo ad accettazione verrà consegnato su un supporto CD-ROM/DVD; vi si allegherà l'Utilità d'installazione, Manuali d'uso ed eventualmente di collaudo, con tutti i sorgenti completi e l'Utilità di compilazione.

2.1.6 Studio di Fattibilità

A seguito della presentazione ufficiale di ciascun capitolato d'appalto$_{\mathbf{G}}$ avvenuta in data venerdì 2019-11-15 ad opera di ogni proponente, il *Responsabile di Progetto* provvede a organizzare delle riunioni tra i membri del gruppo *ProApes* così da incoraggiare un costruttivo scambio di opinioni sui capitolati proposti. Individuato il capitolato d'interesse, ogni *Analista* provvederà a svolgere un'ulteriore attività di analisi con la redazione dello *Studio di Fattibilità*. Tale documento contiene tutte le motivazioni che portano il gruppo a proporsi come fornitore per il prodotto indicato. Per ogni capitolato, tale documento riporta anche:

- **informazioni generali**: un elenco delle informazioni di base quali il nome del progetto, del proponente e dei committenti;
- **descrizione del capitolato**: una sintesi del prodotto da sviluppare secondo quanto descritto dal capitolato d'appalto;

- **finalità del progetto**: le finalità richieste dal capitolato d'appalto;
- **tecnologie interessate**: le tecnologie da impiegare nello svolgimento del progetto;
- **aspetti positivi:** i fattori a favore emersi durante l'analisi preliminare svoltasi sfruttando le conoscenze dei singoli ed effettuando ricerche web e cartacee;
- **criticità e fattori di rischio:** analogo al punto precedente ma per quanto concerne gli aspetti negativi individuati;
- **conclusioni:** l'accettazione o il rifiuto del capitolato, tenendo conto non solo dei fattori sopra espressi, ma anche dell'interesse del gruppo verso la realizzazione del prodotto.

2.1.7 Preparazione al collaudo del prodotto

Una volta raggiunto un prodotto software sufficientemente stabile e completo, i membri del team si impegneranno ad instaurare un rapporto il più possibile costante con l'azienda proponente *Zucchetti S.p.A.*.
Per poter avere esito positivo in sede di collaudo il prodotto finito deve essere stato preventivamente preparato attraverso il superamento di un'attività di test. Questa si articola in:

- **test d'unità**;
- **test di regressione**;
- **test d'integrazione**;
- **test di sistema**.

Solo quando il software avrà superato con successo i test, il gruppo *ProApes* avrà una garanzia di aver realizzato un prodotto affidabile, corretto e completo.
Durante il collaudo va dimostrato al committente che:

- i test presenti nel *Piano di Qualifica 4.0.0-1.10* sono stati eseguiti e hanno ottenuto un esito conforme alle metriche$_{\mathbf{G}}$ dichiarate;
- sono state rispettate le aspettative e le promesse fatte al proponente, andando ad implementare al meglio i requisiti obbligatori definiti nell'*Analisi dei Requisiti 4.0.0-1.10*;
- i requisiti desiderabili e facoltativi, nel caso fossero stati implementati, vanno oltre alle aspettative minime del proponente.

I test sopracitati vengono definiti con maggiore dettaglio in §3.4.8.

2.1.8 Collaudo e consegna

Il gruppo *ProApes* consegnerà all'azienda proponente e ai committenti *Prof. Tullio Vardanega* e *Prof. Riccardo Cardin* quanto segue:

- **Codice sorgente**;
- **Documentazione di prodotto**, ovvero quanto citato in §2.1.5, alla quale si andranno ad aggiungere:
  - ***Glossario 4.0.0-1.10*:** facilita la comprensione dei documenti eliminando le possibili ambiguità inerenti alla terminologia impiegata;
  - **Manuale Utente (da redigere):** guida l'utente finale durante l'installazione e l'utilizzo delle funzionalità del prodotto;
  - **Manuale dello Sviluppatore (da redigere):** permette a sviluppatori esterni di contribuire al progetto agevolmente.

2.1.9 Strumenti

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

Microsoft Excel

Software rilasciato da Microsoft Office specializzato nella realizzazione di fogli elettronici, permette di ottenere in modo semplice:

- diagrammi, istogrammi e areogrammi;
- tabelle e grafici;
- calcoli.

GanttProject

Per semplificare il lavoro del *Responsabile di Progetto* durante il processo di Pianificazione, l'assegnazione delle risorse, la verifica del rispetto dei tempi, la gestione dei budget e l'analisi dei carichi di lavoro, vengono impiegati i diagrammi di Gantt$_{\mathbf{G}}$, generati utilizzando *GanttProject*.

2.2 Sviluppo

2.2.1 Scopo

Lo scopo del processo di Sviluppo, come stabilito dallo standard$_{\mathbf{G}}$ ISO/IEC 12207:1995, è descrivere i compiti e le attività da svolgere, relative al prodotto software da sviluppare.

2.2.2 Descrizione

Di seguito sono elencate e successivamente trattate le varie attività caratterizzanti tale processo:

- **Analisi dei requisiti**;
- **Progettazione architetturale;**
- **Codifica del software**.

2.2.3 Aspettative

Le attese, riguardo il processo in questione, sono:

- stabilire gli obiettivi di sviluppo;
- stabilire i vincoli tecnologici;
- stabilire i vincoli di design;
- realizzare un prodotto finale che superi i test, che soddisfi i requisiti e le richieste del proponente.

2.2.4 Analisi dei Requisiti

2.2.4.1 Scopo

Ricade su ciascun *Analista* il compito di redigere il documento di *Analisi dei Requisiti* individuando i requisiti diretti e indiretti, impliciti ed espliciti che il proponente richiede per la realizzazione del prodotto.
Le finalità dei requisiti sono:

- descrivere lo scopo del lavoro;
- fornire ai *Progettisti* riferimenti precisi ed affidabili;
- fissare le funzionalità e i requisiti concordati col cliente;
- fornire una base per raffinamenti successivi, garantendo un miglioramento continuo del prodotto e del processo di sviluppo;
- facilitare le revisioni del codice;
- fornire ai *Verificatori* riferimenti per l'attività di controllo dei test;
- tracciamento$_{\mathbf{G}}$ di riferimenti sulla mole di lavoro per poterne eseguire una stima dei costi.

2.2.4.2 Descrizione

Grazie all'impiego di un approccio investigativo$_{\mathbf{G}}$ i requisiti possono essere ricavati da varie fonti, tra cui:

- **Capitolati d'Appalto**: il requisito$_{\mathbf{G}}$ è individuato a seguito dalla lettura del capitolato esposto dalla *Zucchetti S.p.A.*;
- **Verbali Interni**: il requisito emerge dalle riunioni interne effettuate dagli *Analisti* del gruppo;
- **Verbali Esterni**: il requisito è ottenuto a seguito di contatti e discussioni con il responsabile aziendale *Dott. Gregorio Piccoli*; ad ogni requisito viene assegnato un codice presente nella tabella dei tracciamenti;
- **Casi d'uso**: il requisito è estrapolato da uno o più casi d'uso; viene riportato il codice univoco del caso d'uso.

2.2.4.3 Aspettative

Ci si attende, dall'attività di Analisi, la produzione della documentazione formale contenente tutti i requisiti richiesti dal proponente.

2.2.4.4 Struttura

L'Analisi dei Requisiti deve possedere una struttura contraddistinta da:

- **Descrizione**: contenente a sua volta le informazioni riguardanti:
  - il prodotto;
  - le principali componenti presenti all'interno del sistema;
  - la piattaforma d'esecuzione;
  - i vincoli di progettazione individuati.
- **Casi d'uso**: tale sezione individua gli attori che interagiscono con le varie componenti del sistema, nonché le interazioni che si scatenano tra il sistema, gli attori e gli elementi esterni al sistema. Viene utilizzato il linguaggio naturale, come mostrato in §2.2.4.6, corredata da relativo diagramma UML 2.0$_{\mathbf{G}}$;
- **Requisiti**: presenza di una tabella contenente:
  - il tracciamento dei requisiti con la dichiarazione di tutti i requisiti obbligatori, desiderabili e opzionali individuati nel sistema durante l'attività di Analisi; anche questi devono essere normati secondo quanto citato in §2.2.4.5;
  - il tracciamento fonte-requisiti, requisiti-fonte, necessario per un riferimento chiaro e formale dell'origine di ogni requisito dichiarato.

2.2.4.5 Classificazione dei requisiti

La convenzione scelta per la rappresentazione dei requisiti è la seguente:

R[Tipologia][Importanza][Codice]

- **Tipologia:** rappresenta una tipologia di requisito, può assumere uno fra i seguenti valori letterali:
  - **V**: requisito di *Vincolo*, descrive vincoli sui servizi offerti dal sistema;
  - **F**: requisito *Funzionale*, descrive servizi o funzioni offerti dal sistema;
  - **P**: requisito *Prestazionale*, descrive i vincoli sulle prestazioni che bisogna soddisfare, come il numero di informazioni che devono essere manipolate durante un certo intervallo di tempo;
  - **Q**: requisito di *Qualità*, descrive i vincoli di qualità da realizzare[3];
- **Importanza**: indica l'importanza associata al requisito, anch'essa viene specificata da uno fra i seguenti valori letterali:
  - **O**: requisito *Obbligatorio*, la sua soddisfazione dovrà necessariamente avvenire per garantire le attività di base del sistema;
  - **D**: requisito *Desiderabile*, non vincola il funzionamento del sistema, ma se implementata fornisce maggiore completezza. Fa parte della categoria di requisiti negoziabili tra *ProApes* e *Zucchetti S.p.A.*;
  - **F**: requisito *Facoltativo*, se soddisfatto rende il sistema più completo, portando, con molta probabilità, ad un dispendio di risorse con conseguente aumento dei costi.
- **Codice**: identificativo univoco del requisito espresso in forma gerarchica padre/figlio. Nello specifico verrà seguito il formalismo:

  [CodiceBase](.[CodiceSottoCaso])*

  dove
  - **CodiceBase**: codice che, combinato alla Tipologia, funge da identificatore del caso d'uso generico generatore del caso d'uso in esame;
  - **CodiceSottoCaso**: codice progressivo opzionale, identificatore degli eventuali sottocasi. Può a sua volta includere altri livelli.

Il *Codice* stabilito secondo la convenzione sopra citata, una volta associato ad un requisito, non potrà cambiare nel tempo.
Ogni requisito deve inoltre essere accompagnato da una serie di informazioni aggiuntive:

- **Descrizione**: descrizione breve, completa e il meno ambigua possibile riguardo lo scopo del requisito;
- **Classificazione**: indicazione dell'importanza del requisito, distinguendo tra Obbligatorio, Desiderabile e Facoltativo; sebbene possa risultare ridondante, facilita la lettura;
- **Fonte**: indicazione della fonte del requisito, individuato tra: *Capitolato d'Appalto, Verbali Interni, Verbali Esterni* e casi d'uso.

[3]Per una descrizione dettagliata sui vincoli di qualità presi in considerazione si veda il documento *Piano di Qualifica 4.0.0-1.10* in §2: *Obiettivi e metriche di qualità*

Requisito	Descrizione	Classificazione	Fonti
RVO1	Autenticazione da parte dell'utente	Obbligatorio	Capitolato UC1

Tabella 3: Esempio di classificazione di un requisito

2.2.4.6 Classificazione dei casi d'uso

La convenzione prescelta per la rappresentazione dei casi d'uso è la seguente:

UC[CodiceBase](.[CodiceSottoCaso])*

Composta da:

- **UC:** acronimo di "use case";
- **CodiceBase:** identificativo del caso d'uso generico;
- **CodiceSottoCaso:** identificativo opzionale per gli eventuali sotto casi del caso d'uso.

Ogni caso d'uso possiede inoltre la seguente struttura:

- **Identificativo:** codice del caso d'uso prescelto secondo quanto stabilito dalla convenzione enunciata sopra;
- **Nome:** stringa testuale relativa al titolo del caso d'uso posta immediatamente dopo all'identificativo;
- **Descrizione grafica:** opzionale, realizzata impiegando il formalismo dell'UML 2.0[4]; rappresenta graficamente i casi d'uso facilitandone la comprensione (sistema, attori, azioni);
- **Descrizione:** breve descrizione del caso d'uso;
- **Attori:** un attore è tutto ciò che è esterno al sistema e con il quale esso interagisce. Possono essere individuate due tipologie di attori:
  - **Attore Primario:** interagisce attivamente con il sistema per il raggiungimento di un obiettivo specifico;
  - **Attore Secondario:** entità esterna al sistema che aiuta l'attore primario nel raggiungimento del suo scopo.
- **Precondizione$_G$:** specifica le condizioni del sistema prima del verificarsi degli eventi del caso d'uso;
- **Input:** opzionale, consiste in un valore o in un oggetto portato dall'attore all'interno del sistema. Il suo impiego è previsto nei casi dove si voglia specificare con precisione le informazioni che il sistema riceve in input;
- **Postcondizione$_G$:** specifica le condizioni del sistema dopo il verificarsi degli eventi del caso d'uso;
- **Output:** opzionale, consiste in un valore o in un oggetto risultante come conseguenza di un'azione eseguita dall'attore;
- **Scenario principale:** rappresentazione attraverso elenco numerato del flusso degli eventi;
- **Scenario alternativo:** opzionale, rappresentazione attraverso elenco numerato di tutte le azioni che possono manifestarsi durante l'esecuzione di un caso d'uso per un evento imprevisto che causa la deviazione dallo scenario principale;
- **Estensioni:** opzionali, impiegati per modellare scenari alternativi. Al verificarsi di una determinata condizione, il caso d'uso ad essa collegata viene interrotto;

[4] Lo strumento da utilizzare è specificato nell'apposita sotto-sezione in §3.1.8.5

- **Inclusioni:** opzionali, usati quando due casi d'uso sono tra loro collegati. Il secondo è incondizionatamente incluso nel primo;
- **Generalizzazioni:** opzionali, rappresentano specializzazioni di un caso d'uso.

Di seguito viene riportato un esempio di caso d'uso:

UC1 - Autenticazione

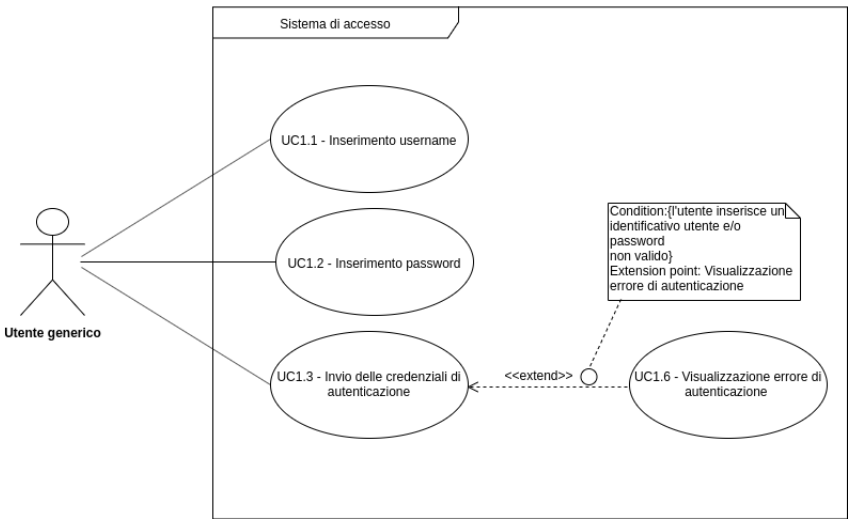

Figura 1: Descrizione grafica del caso d'uso

- **Descrizione:** L'utente non autenticato effettua il login nel sistema.
- **Attori:** Utente generico.
- **Precondizione:** L'utente non è ancora registrato presso il sistema.
- **Input:** Insieme di valori rappresentanti l'username e la password scelte dall'utente.
- **Postcondizione:** L'utente è in possesso delle credenziali per accedere al sistema.
- **Scenario Principale:**
  1. L'utente entra nel sistema;
  2. L'utente provvede a registrarsi presso il sistema selezionando la funzionalità "Registrati";
  3. L'utente inserisce un'username univoco nel sistema (U.C1.1);
  4. L'utente inserisce una password coerente coi vincoli imposti (U.C1.2);
  5. L'utente effettua l'invio delle credenziali di accesso (U.C1.3).
- **Estensioni:**
  1. L'utente inserisce un'username già presente nel sistema:
     (a) Fallisce la registrazione dell'utente presso il sistema;
     (b) Viene mostrato un messaggio d'errore di username o password errata (U.C1.6).

2.2.4.7 Qualità dei requisiti

Ciascun requisito dovrà rispettare le seguenti qualità:

- **completezza:** descrizione specifica e dettagliata di ogni funzionalità richiesta dal prodotto software e del suo comportamento in risposta agli input dati;
- **consistenza:** non devono crearsi situazioni di contraddizione tra requisiti;
- **correttezza:** il requisito deve risultare veramente necessario e richiesto dagli utenti finali;
- **univocità:** per evitare situazioni di ambiguità, ogni requisito deve essere contraddistinto da un codice formale univoco;
- **verificabilità:** si deve poter verificare che il sistema realizzi il requisito individuato;
- **modificabilità:** il requisito deve poter cambiare nel tempo rimanendo consistente e completo;
- **tracciabilità:** di un requisito bisogna avere chiara l'origine e deve essere possibile referenziarla in futuro.

2.2.5 Progettazione

2.2.5.1 Scopo

L'attività di Progettazione individua le caratteristiche che il prodotto software deve possedere per fornire la soluzione migliore che soddisfi i requisiti degli stakeholders$_{\mathbf{G}}$, come esposto nel documento *Analisi dei Requisiti 4.0.0-1.10*. Essa deve provvedere a:

- garantire, applicando un approccio sistematico$_{\mathbf{G}}$ ai problemi, la qualità del prodotto sviluppato, perseguendo un principio di correttezza fatto in modo costruttivo;
- organizzare e suddividere compiti implementativi, provvedendo così a diminuire la complessità del problema originale fino alle singole componenti, facilitandone la codifica da parte dei *Programmatori*;
- ottimizzare i tempi e il numero delle risorse assegnate.

2.2.5.2 Descrizione

Tale attività punta a realizzare l'architettura del sistema. Inizialmente realizzata dal *Proof of Concept* della *Technology Baseline*, poi approfondita e descritta nel documento tecnico allegato alla *Product Baseline*.

2.2.5.3 Aspettative

Prima di procedere alla realizzazione architetturale, il gruppo dovrà:

- definire le tecnologie da utilizzare;
- approfondire e studiare gli aspetti positivi e le eventuali criticità;
- produrre una bozza dimostrabile del prodotto, chiamata *Proof of Concept*, che rifletta l'approfondimento svolto.

Durante le attività di Progettazione, l'architettura del sistema deve essere realizzata rispettando ogni vincolo stabilito con l'azienda *Zucchetti S.p.A.*.

2.2.5.4 Qualità dell'architettura

È compito di ogni *Progettista* definire un'architettura logica del prodotto di qualità. Le varie componenti devono essere identificabili in modo chiaro, riusabile e coeso e garantendo di rimanere entro i costi fissati.
Per assicurare quanto appena espresso l'architettura dovrà presentare le seguenti caratteristiche:

- soddisfare i requisiti indicati nel documento *Analisi dei Requisiti 4.0.0-1.10* e adattarsi facilmente nel caso essi evolvano o se ne aggiungano di nuovi;
- rispettare le caratteristiche di comprensibilità, modularità e robustezza riuscendo a gestire situazioni erronee improvvise;
- risultare affidabile, essere quindi in grado di svolgere i compiti cui è destinata ogni volta ne sia richiesto l'utilizzo, anche nel caso di mancanze momentanee;
- rispettare le caratteristiche di safety$_{\mathbf{G}}$ e security$_{\mathbf{G}}$ rispetto ai malfunzionamenti;
- essere disponibile, ovvero presentare poco tempo di indisponibilità totale a fini manutentivi;
- presentare un impiego efficiente delle risorse necessarie;
- garantire una riusabilità, ovvero un impiego delle sue parti anche in altre applicazioni;
- presentare componenti semplici, coese nel raggiungimento degli obiettivi, incapsulate e con un basso livello di accoppiamento.

2.2.5.5 Periodi della Progettazione

Le attività da svolgere durante il progetto vengono suddivise per obiettivi da raggiungere e fatte rientrare all'interno di appositi periodi temporali, definiti nel *Piano di Progetto 4.0.0-1.10*. Come esposto di seguito, essi consistono nella Progettazione architetturale e nella Progettazione di dettaglio e Codifica, durante i quali verranno affrontate la *Technology Baseline* e la *Product Baseline*.

Progettazione architetturale

Durante questo periodo vengono definite le componenti del sistema, ovvero i task da svolgere per portare a termine positivamente il periodo:

- individuazione delle componenti coinvolte nel sistema;
- definizione dei ruoli e delle responsabilità per ogni componente;
- definizione delle interazioni tra componenti;
- creazione e documentazione, all'interno del *Piano di Qualifica 4.0.0-1.10* dei test d'integrazione. Tali test accertano che le componenti del sistema, una volta integrate assieme, abbiano un comportamento adeguato.

Progettazione di dettaglio

Estensione di quanto svolto durante la Progettazione architetturale. Si scompongono le parti fino a specificare cosa dovrà fare il codice nel particolare, sarà così possibile assegnare compiti indivisibili ai *Programmatori*:

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

2.2.5.6 Design pattern

Al termine dell'attività di Analisi dei Requisiti sarà sempre compito dei *Progettisti* adottare opportune soluzioni progettuali a problemi ricorrenti, riportandoli nel documento di *Technology Baseline*. Tali soluzioni, denominate design pattern, si distinguono per:

- flessibilità;
- capacità di garantire una discreta libertà d'uso ai *Programmatori*.

Ogni design pattern$_{\mathbf{G}}$ deve essere accompagnato da:

- un diagramma che lo illustri;
- una descrizione testuale del suo funzionamento;
- descrizione dell'utilità di tale design pattern all'interno dell'architettura realizzata.

2.2.5.7 Diagrammi UML 2.0

Per dare maggiore chiarezza alle scelte progettuali adottate e ridurre le eventuali ambiguità che possono crearsi, si predispone l'utilizzo dei diagrammi UML 2.0.
In particolare sono previsti:

- **diagrammi delle attività:** descrittivi graficamente del flusso di operazioni di un'attività attraverso una logica procedurale; impiegati per descrivere, ad esempio, la logica di un algoritmo;
- **diagrammi delle classi:** illustrativi graficamente di una collezione di elementi di un modello; astraendosi da un linguaggio di programmazione specifico definiscono le classi, i tipi, i metodi, gli attributi e le relazioni che vi intercorrono;
- **diagrammi dei package$_{\mathbf{G}}$:** rappresentativi graficamente di raggruppamenti di classi in unità; esse devono venire riusate assieme e condividono la stessa causa di cambiamento;
- **diagrammi di sequenza:** illustrativi graficamente di sequenze di azioni che avvengono tra oggetti/classi attraverso l'impiego di scelte definite; per esempio, possono rappresentare una sequenza di invocazioni tra metodi.
- **diagrammi dei casi d'uso:** rappresentativi graficamente in modo dettagliato delle funzionalità del sistema offerte al proponente.

Si potrà inoltre fare uso di altre tipologie di diagrammi, come diagrammi delle componenti, di macchine a stati o di deployment$_{\mathbf{G}}$, nel caso ve ne fosse necessità.

Diagrammi delle attività

Un diagramma delle attività definisce le caratteristiche dinamiche del sistema sia per attività e concetti legati all'implementazione dei metodi, che a livelli di astrazione più elevati. Un diagramma descrive un processo, ovvero una sequenza di elementi azioni/attività collegate tra loro da frecce. Viene impiegato il formalismo seguente:

- **nodo iniziale:** in ogni diagramma vi deve sempre essere un nodo d'inizio, rappresentato da un pallino pieno. Punto d'inizio dell'esecuzione, viene generato un token$_{\mathbf{G}}$;

Figura 2: Nodo iniziale di un diagramma delle attività

- **nodo finale di flusso:** rappresentato con una X contenuta in un cerchio vuoto; indica un punto in cui uno dei rami dell'esecuzione termina; il token viene consumato, tuttavia l'esecuzione non è da ritenersi conclusa: può continuare su altri rami paralleli a quello terminato;

Figura 3: Nodo finale di flusso di un diagramma delle attività

- **nodo finale:** ad ogni nodo iniziale deve corrispondere uno finale; rappresentato da due cerchi concentrici, di cui il più interno pieno e quello esterno vuoto, rappresenta il punto in cui viene consumato un token;

Figura 4: Nodo finale di un diagramma delle attività

- **Activity:** rappresenta un'attività attraverso un rettangolo smussato contenente la descrizione della stessa; quest'ultima assume un ruolo di carattere generale, è breve e concisa, composta da parole chiavi, la prima delle quali inizia con la lettera maiuscola; consuma e produce il token;

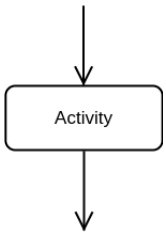

Figura 5: Activity di un diagramma delle attività

- **Subactivity:** rappresenta una sottoattività attraverso un rettangolo smussato contenente la descrizione della stessa; impiegata quando una singola attività ne modella di più.
  Per evitare la creazione di diagrammi di dimensione troppo elevata, che ne comporterebbero una gestione difficoltosa, l'Activity (attività principale), in basso a destra presenta un piccolo tridente, usato come riferimento ad una sottattività permettendone così una trattazione a parte. Ogni sottoattività è composta da un input ed un output e viene contornata da un Activity frame;

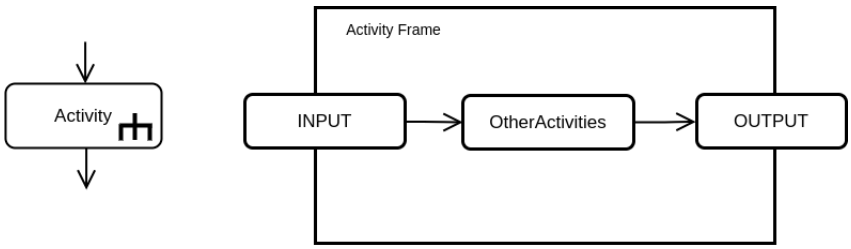

Figura 6: Subactivity di un diagramma delle attività

- **Fork:** rappresentato da una singola freccia entrante e da n frecce uscenti, tra le quali viene posta una lunga linea orizzontale o verticale, a seconda dell'orientazione; è il punto dove l'attività si parallelizza senza alcun vincolo di esecuzione temporale; consuma un token e ne genera n;
- **Join:** rappresentato da n frecce entranti e una sola freccia uscente, tra le quali viene posta una lunga linea orizzontale o verticale, a seconda dell'orientazione, è il punto dove avviene la sincronizzazione tra processi paralleli; consuma n token e ne genera uno;

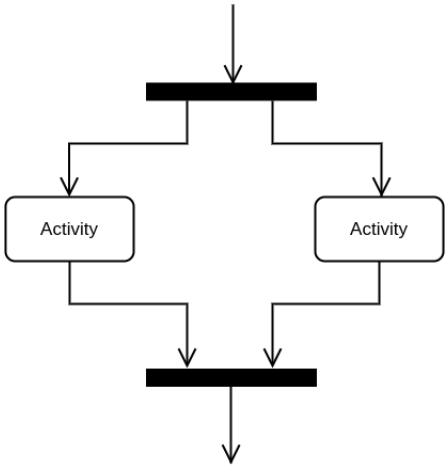

Figura 7: Fork e Join di un diagramma delle attività

- **Branch:** rappresentato da un rombo con una freccia entrante ed n uscenti; ad ognuna di esse vengono associate delle guardie, indicate nel formato `[condizione guardia]`, ovvero delle condizioni che permettono ai token di proseguire per un solo ramo; non viene generato alcun token, ma viene instradato;
- **Merge:** rappresentato da un rombo con n frecce entranti ed una freccia uscente; è il punto dove viene chiusa la parte condizionale, gli n rami generati dal Branch tornano ad unirsi; non viene né consumato né prodotto alcun token, in questo caso viene solo instradato;

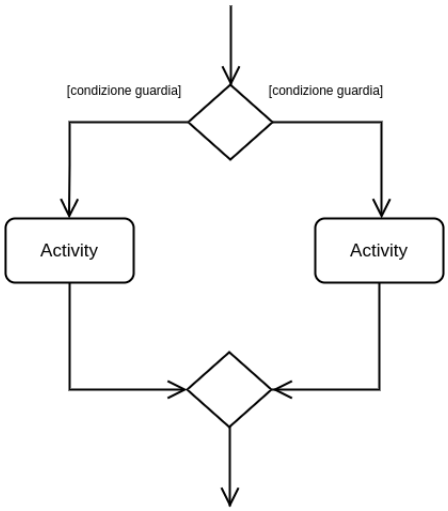

Figura 8: Branch e Merge di un diagramma delle attività

- **Pin:** rappresentato da un piccolo quadratino, da dove entrano/escono frecce dalle Activity; funge da indicatore del passaggio di un parametro, il cui tipo va indicato a fianco seguendo il formato `tipoParametro`;

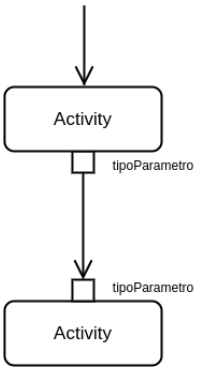

Figura 9: Pin di un diagramma delle attività

- **Segnali:** rappresentati mediante due figure "a incastro": la prima non bloccante per l'emissione del segnale, la seconda invece bloccante per la ricezione dello stesso; all'interno delle figure devono essere contenute le diciture *«signal sending»* e *«signal receipt»* rispettivamente, ognuna delle quali deve essere seguita da una descrizione breve e concisa del segnale che rappresenta;

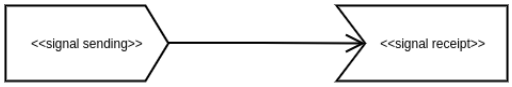

Figura 10: Segnali di un diagramma delle attività

- **Timeout:** rappresentato da una clessidra, modella timeout o eventi ripetuti; i timeout sono composti da frecce entranti ed uscenti, invece gli eventi ripetuti presentano solo archi uscenti, sotto vi deve essere una descrizione così composta: i timeout devono iniziare con la parola chiave "wait", gli eventi ripetuti invece con "every"; in ambedue le circostanze la parola chiave dovrà essere seguita da un indicatore temporale o da una descrizione, usando un linguaggio naturale breve e conciso.

Figura 11: Timeout di un diagramma delle attività

Diagrammi delle classi

Un diagramma delle classi definisce le caratteristiche statiche di un sistema così da descrivere ciò che non riguarda l'ambiente a run-time.
Ogni classe viene strutturata come un rettangolo tripartito orizzontalmente e a partire dall'alto presenta le seguenti tre suddivisioni:

1. **Nome della classe**: è l'identificatore della classe stessa; scritto in inglese, a lettere maiuscole e in grassetto, deve essere evocativo del compito svolto. Nel caso di una classe *astratta* il nome deve essere presentato in formato italico, nel caso di un'*interfaccia* deve sempre venire preceduto dalla direttiva `<<interface>>`;
2. **Attributi**: presentati l'uno dopo l'altro ciascuno in una riga a sé, devono seguire il formato:

   Visibilità nome: tipo [molteplicità ordinamento] = valori di default

   - **Visibilità**: ogni attributo dovrà essere preceduto obbligatoriamente da uno dei seguenti indicatori:
     - **-**: indicatore di visibilità privata;
     - +: indicatore di visibilità pubblica;
     - #: indicatore di visibilità protetta;
     - ∼: indicatore di visibilità di package.
   - **Nome**: ogni attributo deve essere univoco, significativo ed espresso nel formato `nomeVariabile: tipo`. Se la variabile è di tipo *costante* il nome deve essere tutto in maiuscolo, rispettando il formato `NOMEVARIABILE: tipo`. Il tipo può essere anche definito dall'utente;
   - **Molteplicità**: nel caso di un numero multiplo di elementi (e.g: liste o array), se ne può definire il numero esatto; il formato da usare è `tipoVariaibile[molteplicità]`. In particolare * rappresenta una molteplicità non conosciuta a priori, nel caso invece di un singolo elemento la sua dichiarazione può venire omessa;
   - **Ordinamento**: se vi è la necessità di far seguire all'attributo anche l'informazione sull'ordinamento seguito, per collezioni in cui vi sia un ordine si utilizza la proprietà *orderd*; in caso contrario si fa ricorso ad *unordered*. Per la sua dichiarazione si usa la forma `tipoVariaibile[molteplicità ordinamento]`;
   - **Valori di default**: per ogni attributo presentato possono venire indicati anche valori di default.

   Nel caso di tipi non primitivi va preferito l'uso di *tipi composti*: classi ad hoc in cui definire tutti i tipi che compongono il tipo principale e collegate alla classe di partenza mediante una freccia unidirezionale.
   È possibile usare dei *commenti*; ogni commento al suo interno può racchiudere un altro commento.
3. **Metodi**: descrittori del comportamento di una classe (non dell'implementazione), vanno presentati uno di seguito all'altro, ciascuno occupante una specifica riga; vanno dichiarati secondo il formato:

   Visibilità nomeMetodo (lista-paramentri-formali): return-type

- **Visibilità**: come per gli attributi anche nel caso dei metodi, questi devono venire preceduti da un indicatore di visibilità;
- **Nome del metodo**: ogni metodo deve essere univoco, significativo, espresso in inglese e scritto con la prima lettera minuscola rispettando la notazione di tipo CamelCase$_{\mathbf{G}}$ per nomi composti;
- **Lista dei parametri formali**: contenete da 0 fino ad n parametri separati da una virgola, ciascuno dei quali segue le medesime regole imposte per gli attributi;
- **Return-type**: rappresenta il tipo di ritorno dell'operazione.

Se un metodo è *astratto* va scritto in italico, se invece si presenta come *statico* va sottolineato.
Nella lista dei metodi possono non essere inclusi i metodi dei costruttori, i metodi getter e setter semplici, a meno di apposita direttiva di un *Progettista*.
Anche i metodi possono venire accompagnati da *commenti*.

Se in una classe variabili e/o metodi fossero assenti, nel diagramma apparirà comunque, seppur vuota, la sezione ad esse dedicata.

I diagrammi delle classi sono collegati fra loro da frecce che ne esplicitano le *dipendenze*. In particolare, verranno utilizzati i seguenti tipi di freccia:

- **Dipendenza**: freccia tratteggiata da una classe A ad una classe B; rappresenta il minimo grado di dipendenza fra le classi e significa che A dipende da B in base ad una primitiva posta sopra la freccia. Le primitive possibili sono:
  - **«call»**: A invoca un metodo di B;
  - **«create»**: A crea istanze di B;
  - **«derivate»**: A deriva da B;
  - **«instantiate»**: A è un'istanza della classe B;
  - **«permit»**: B permette ad A di accedere ai suoi campi privati;
  - **«realize»**: A è un implementazione di una specifica o di un'interfaccia definita da A;
  - **«refine»**: indica un raffinamento tra differenti livelli semantici;
  - **«substitute»**: A si può sostituire con B;
  - **«trace»**: tiene traccia dei requisiti o di come i cambiamenti di una parte di modello si colleghino ad altre;
  - **«use»**: A richiede B per la sua implementazione.

Figura 12: Relazione di dipendenza in un diagramma delle classi

- **Associazione**: freccia semplice da una classe A ad una classe B, significa che A contiene dei campi dati o delle istanze di B. Le molteplicità possibili indicate agli estremi della freccia sono:
  - **1**: A possiede un'istanza di B;
  - **0..1**: A possiede 0 o 1 istanze di B;
  - **0..***: A possiede 0 o più istanze di B;
  - *****: A possiede più istanze di B;

- **n**: A può possedere n istanze di B.

Figura 13: Relazione di associazione in un diagramma delle classi

- **Aggregazione**: freccia a diamante vuota, da una classe A ad una classe B, significa B è "parte di" A (relazione part of); A non ha senso di esistere senza B. Gli aggregati possono essere condivisi;

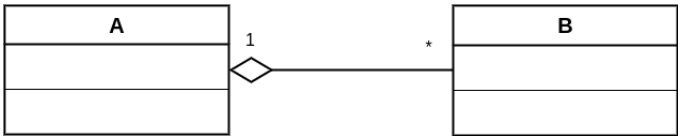

Figura 14: Relazione di aggregazione in un diagramma delle classi

- **Composizione**: freccia a diamante piena, si tratta di un'aggregazione ancora più forte di quanto descritto al punto sopra. Da una classe A ad una classe B indica che:
  - le due classi devono venire usate sempre assieme;
  - gli aggregati possono appartenere ad un solo aggregato (aggregato con cardinalità (1,1));
  - solo l'oggetto intero può creare e distruggere le sue parti.

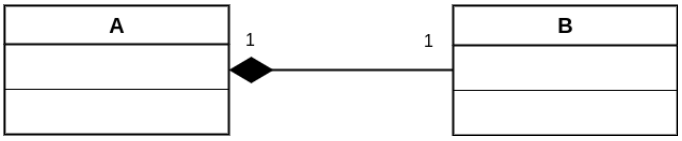

Figura 15: Relazione di composizione in un diagramma delle classi

- **Generalizzazione/Ereditarietà**: freccia vuota continua, da una classe A ad una classe B, è il massimo grado di dipendenza fra le classi, soprattutto se si estende un tipo concreto. Indica "che un oggetto di A è anche un oggetto di B";

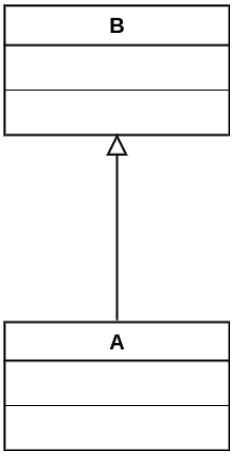

Figura 16: Relazione di generalizzazione in un diagramma delle classi

- **Subtyping**: usata per implementare un'intefaccia o una classe astratta; se un'interfaccia B viene implementata con una classe (astratta o concreta) A, questo viene rappresentato per mezzo di una freccia tratteggiata da A a B.

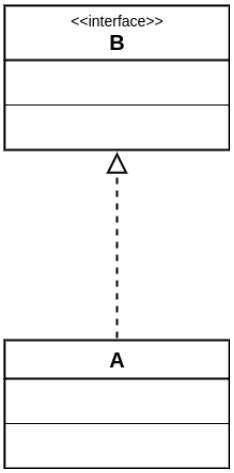

Figura 17: Relazione di implementazione in un diagramma delle classi

Diagrammi dei package

Un diagramma di package permette la definizione di un livello più al dettaglio dell'architettura. Ciò viene attuato attraverso il raggruppamento di un numero arbitrario di elementi UML, ovvero classi, in una unità di livello più alto: il *package*. Vanno raggruppate solo le classi e ogni classe può appartiene ad un unico package che ne può contenere un altro a sua volta.
Ogni package, identificativo di un namespace$_{\mathbf{G}}$, viene rappresentato attraverso un rettangolo con un'etichetta per il nome, contenente i diagrammi delle classi appartenenti al package ed eventuali sotto-package.
Il nome deve essere completamente qualificato e distinto per ogni elemento. La struttura da rispettare è la seguente:

package::package:: ... ::classe

Ogni elemento in un package può avere *visibilità* pubblica (+) o privata (-).
Le *dipendenze* tra i vari package sono segnalate con una freccia tratteggiata. Tale freccia fatta partire dal package A al package B, indica una dipendenza di A nei confronti di B. Vanno evitare le dipendenze cicliche.

Figura 18: Dipendenza fra packages

Un pakage può contenere anche *interfacce* e/o *classi astratte*; in questo caso si utilizza una freccia vuota continua, dal package che estende a quello che viene esteso.

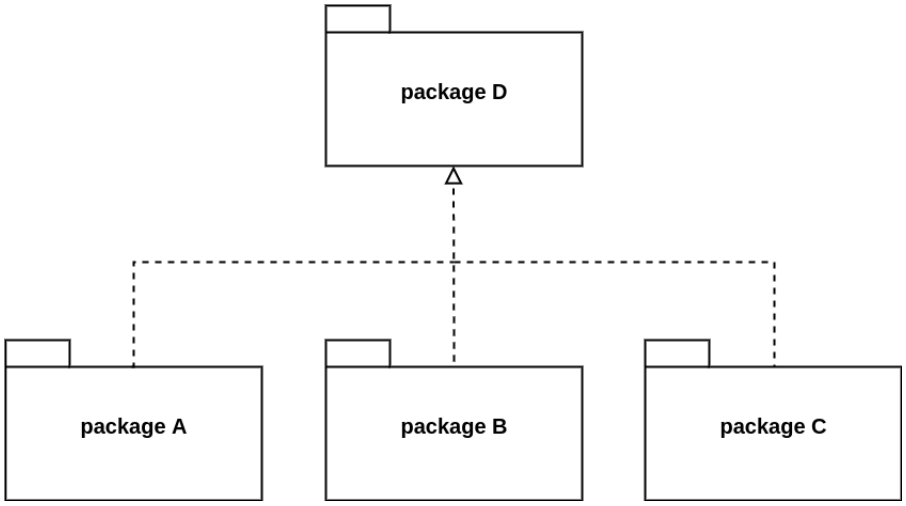

Figura 19: Package di interfaccia

Diagrammi di sequenza

Un diagramma di sequenza descrive formalmente il modo in cui le varie entità (classi o processi), collaborano fra loro; possiede un senso di lettura verticale dall'alto verso il basso (indicatore dello scorrere del tempo), dove lo scenario descritto si presenta come una precisa sequenza di azioni.
Attraverso l'uso di questi diagrammi si è in grado di:

- modellare la logica di procedure, metodi ed operazioni;
- mostrare l'interazione tra le varie componenti coinvolte;
- comprendere e pianificare le funzionalità inerenti uno specifico scenario.

Le istanze coinvolte vengono rappresentate con un rettangolo, al cui interno viene posto il nome necessario per identificarle nel formato `istanza : NomeClasse` in grassetto.
Sotto ad ogni istanza viene collocata una linea tratteggiata rappresentante la sua linea della vita. In alcune punti, quest'ultima, viene sormontata da una *barra di attivazione* che indica i momenti in cui l'entità è effettivamente attiva (viene omessa quando l'entità risulta sempre attiva). Dalla barra di attivazione partono poi delle frecce, rappresentanti un messaggio/segnale, verso entità già istanziate o, in alternativa, verso una nuova istanza di classe per crearla.
In particolare, i tipi di frecce utilizzate sono:

- **Freccia piena per indicare un messaggio sincrono**: rappresenta la chiamata di un metodo. Sopra tale freccia va specificato il metodo invocato, secondo la forma `nomeMetodo(lista-parametri-formali)`. Ci si aspetta un segnale di risposta;
- **Freccia per indicare un messaggio asincrono**: rappresenta gli stessi concetti esposti al punto sopra e fa uso dello stesso formalismo, la differenza sta nel fatto che in questo caso non si attende un segnale di risposta;
- **Freccia tratteggiata sormontata da «create»**: indica la creazione di una nuova entità e termina sempre in un rettangolo che ne contiene il nome, nel rispetto della forma `istanza: NomeClasse`;
- **Freccia piena sormontata da «destroy»**: indica la distruzione di un'entità e si conclude sempre con una X, nella quale muore la linea della vita dell'istanza stessa;
- **Freccia tratteggiata per indicare il ritorno di un metodo chiamato**: rappresenta un segnale di risposta, sopra tale freccia va indicato il tipo di ritorno, nel rispetto della forma `TipoRitorno`.

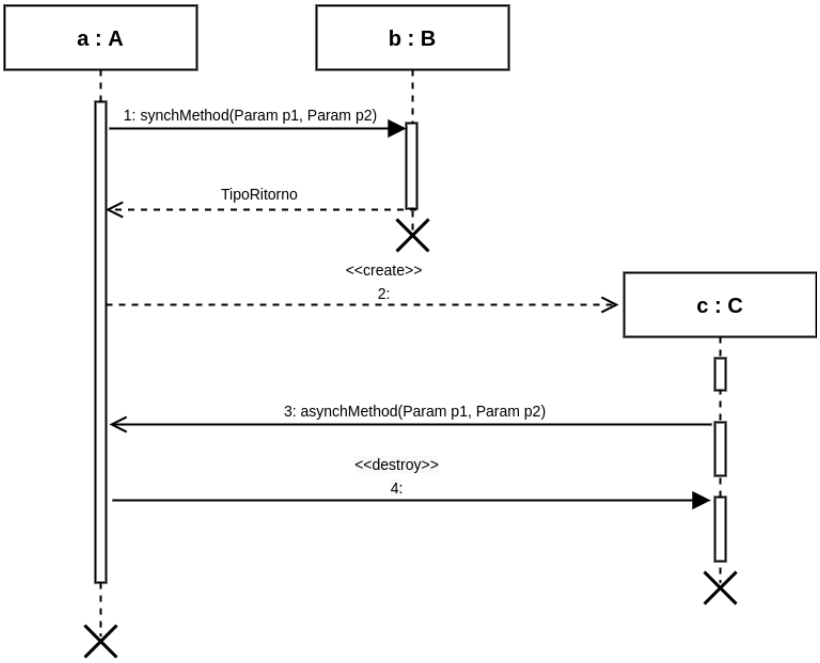

Figura 20: Diagramma di sequenza con tutti i tipi di segnale

È inoltre possibile determinare sui diagrammi di sequenza dei frame di interazione associati ad una guardia (o condizione). I possibili sono:

- **alt**: indica dei frammenti multipli in alternativa fra loro; viene eseguito solo quello per il quale la condizione risulta verificata;
- **opt**: opzionale; indica un frammento che viene eseguito solo se la guardia specificata risulta vera. Equivalente a alt con solo una freccia;
- **par**: parallelo; indica che ogni frammento viene eseguito in parallelo;
- **loop**: indica un ciclo; ogni frammento può venire eseguito più volte e la base dell'iterazione viene indicata dalla guardia;
- **region**: indica una regione critica; il frammento deve essere eseguito in mutua esclusione (un solo thread alla volta);
- **neg**: indica un qualcosa di negativo; il frammento evidenzia un'interazione non valida;
- **ref**: indica un riferimento; ovvero un'interazione definita in un altro diagramma;
- **sd**: *Sequence diagram*; utilizzato per racchiudere un intero diagramma di sequenza.

Diagrammi dei casi d'uso

Un caso d'uso descrive le funzionalità del sistema attraverso una visione esterna. Nello specifico, un caso d'uso è un insieme di scenari e sequenze di azioni con in comune il medesimo 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**: gli attori rappresentano tutto ciò che è esterno al sistema e ci interagisce. Un caso d'uso offre funzionalità all'attore che può essere una persona oppure un altro sistema esterno. Non è concessa la possibilità di definire alcun dettaglio implementativo sui modi di interazione.
  Un attore viene disegnato come un omino stilizzato, sotto viene posto il nome nel rispetto della forma `Nome attore`;

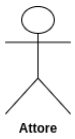

Figura 21: Attore di un caso d'uso

- **Caso d'uso**: rappresentato con un ovale, al suo interno viene inserita la descrizione dello stesso.
  La forma impiegata consiste in una numerazione di tipo `UCx.y`, con a seguire un `"-"` (trattino) e una `breve descrizione del caso d'uso`; tale descrizione deve risultare concisa ma sufficientemente descrittiva dello scenario. Ogni caso d'uso viene associato ad un attore e viceversa per mezzo di una linea semplice.

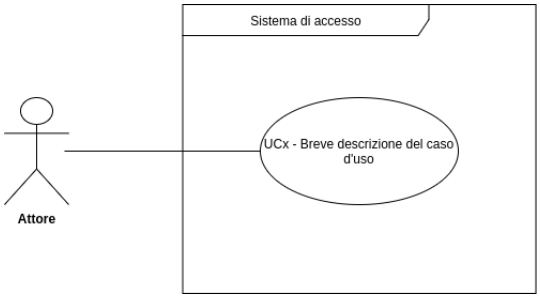

Figura 22: Rappresentazione di un caso d'uso

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

- **Inclusione**: se esiste un'inclusione tra un caso d'uso A e un caso d'uso B, significa che ogni istanza di A deve eseguire anche B. B anche se non ne è a conoscenza, viene perciò incondizionatamente incluso nell'esecuzione di A; in questo modo la responsabilità di esecuzione ricade esclusivamente su quest'ultimo. Tale strategia evita la ripetizione e aumenta il riutilizzo di una medesima struttura, ma va usata solo nei casi che rispettano le condizioni appena dichiarate.
  Un'inclusione viene rappresentata con una freccia tratteggiata, che collega i casi d'uso coinvolti, in direzione del caso d'uso incluso. Viene posta al di sopra della stessa la direttiva `<<include>>`;

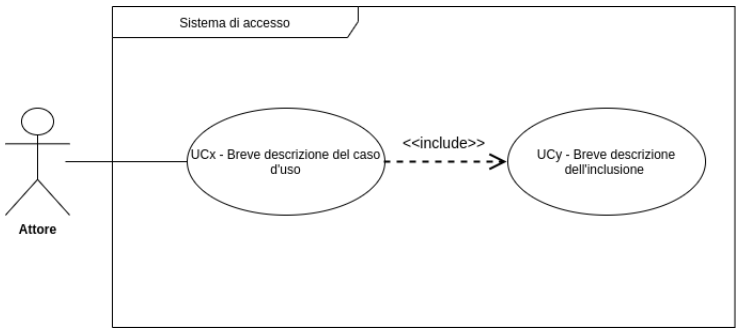

Figura 23: Relazione di inclusione

- **Estensione**: se esiste un'estensione tra un caso d'uso A e un caso d'uso B, significa che ogni istanza di A esegue B in modo condizionato; l'esecuzione di B interrompe A e la responsabilità di esecuzione dei casi di estensione è di chi estende, ovvero B.
  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. Viene posta al di sopra della stessa la direttiva `<<extend>>`, 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. Tale riquadro viene collegato alla freccia mediante una linea tratteggiata.

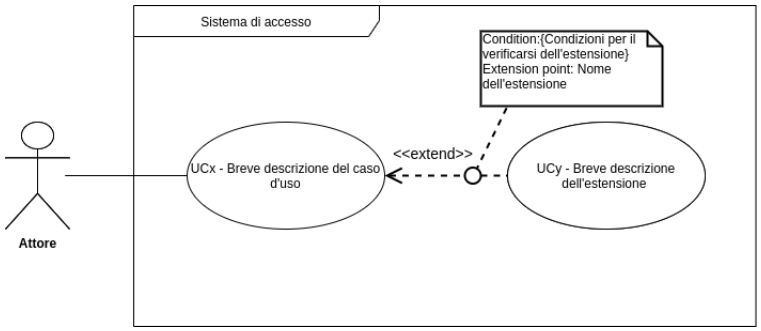

Figura 24: Relazione di estensione

- **Generalizzazione**: la generalizzazione può coinvolgere sia attori che casi d'uso. Nel primo caso se "Utente generico" è generalizzazione di "Admin" significa che quest'ultimo condivide col primo almeno le stesse funzionalità che vi sono espresse. Nei casi d'uso, invece, i figli possono aggiungere funzionalità rispetto ai padri o modificarne il comportamento; tutte le funzionalità non ridefinite nel figli si mantengono in questi come definite nel padre.
  Le generalizzazioni sia di attori che di casi d'uso vengono rappresentate con una freccia continua vuota dal elemento figlio all'elemento padre.

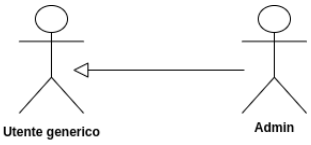

Figura 25: Generalizzazione di attori

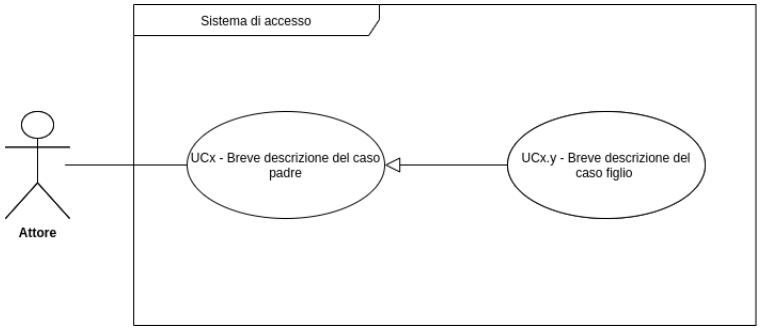

Figura 26: Generalizzazione di un caso d'uso

Un caso d'uso per essere completo, oltre al diagramma deve venire sempre accompagnato da una descrizione testuale dello stesso. Nel caso poi il contesto modellato presenti un'unico caso d'uso, la rappresentazione grafica può venire omessa.

Come il formalismo spiegato in questa sezione venga utilizzato viene presentato dettagliatamente in §2.2.4.6.

2.2.5.8 Test

Ogni *Progettista* avrà inoltre il compito di definire test opportuni, che dovranno essere accompagnati da eventuali classi utili ad individuare errori ed anomalie. I test progettati dovranno rispettare le convenzioni di nomenclatura specificate in §3.5.5.3.

2.2.5.9 Qualità della Progettazione

Il fornitore deve garantire la realizzazione di un prodotto software che soddisfi tutti i requisiti individuati durante l'attività di Analisi: l'attività di Progettazione mira infatti ad ottenere un prodotto di qualità.
Per riuscire ad ottenere quanto appena descritto è vincolante il rispetto dei seguenti punti:

- prima di iniziare qualsiasi attività di Progettazione è necessario avere piena chiarezza delle funzionalità e dei requisiti da soddisfare; ogni attività deve essere coerente con quanto descritto all'interno dell'*Analisi dei Requisiti 4.0.0-1.10*; non possono essere individuate nuove funzionalità da realizzare se non pertinenti all'attività in esame;
- per semplificare il tracciamento dei problemi che possono manifestarsi durante il periodo di Codifica, la suddivisione delle risorse necessarie deve essere il più equa possibile; l'attività d'implementazione deve inoltre risultare sufficientemente realistica: sulla base di tempo/persona disponibile, con scadenze prestabilite e task di lavoro minimi;
- è necessario garantire la correttezza per costruzione.; presentata la progettazione al committente, essa deve essere il più corretta possibile, per premettere appieno l'attuazione del modello di sviluppo incrementale;
- per mantenere accettabile il livello di dipendenza tra le varie componenti del sistema, ciascuna di esse deve essere modulare e con responsabilità singola;
- aderenza agli standard di riferimento con l'uso normato all'interno del presente documento;
- tutte le condizioni minime necessarie riguardanti tempi e strumentazioni utili a produrre, testare, installare e mettere in funzione il prodotto software devono essere assicurate, se questo dovesse mancare, la buona qualità della Progettazione non è da ritenersi soddisfatta.

2.2.6 Codifica

2.2.6.1 Scopo

Con la Codifica, svolta dal ruolo del *Programmatore*, si norma l'effettiva realizzazione del prodotto software richiesto. Permette di trasformare in codice l'architettura ad alto livello pensata dai *Progettisti*, rendendola eseguibile dai calcolatori.
I *Programmatori* dovranno seguire queste norme durante le fasi di programmazione ed implementazione.

2.2.6.2 Descrizione

La scrittura del codice deve rispettare quanto stabilito nella documentazione di prodotto. Dovranno quindi essere perseguiti gli obiettivi descritti all'interno del documento *Piano di Qualifica 4.0.0-1.10*, garantendo la scrittura di codice di qualità.
In questa sezione sono prima trattate le regole di carattere più generale, da adottarsi per ogni linguaggio di programmazione impiegato nello svolgimento del progetto, seguono quelle più specifiche per il linguaggio principale *ECMAScript 6*$_{\boldsymbol{G}}$.
Ogni norma presenta un paragrafo di appartenenza, un titolo, una breve descrizione e, se il caso lo richiede, un esempio.

2.2.6.3 Aspettative

L'attività di Codifica punta ad un prodotto software che rispetti le richieste stabilite col proponente. L'uso di norme e convenzioni in questa fase è fondamentale per:

- garantire la generazione di codice leggibile ed uniforme;
- agevolare le attività di manutenzione, verifica e validazione;
- migliorare la qualità di prodotto.

2.2.6.4 Convenzioni linguistiche

Nomi di variabili, costruttori, metodi, classi e i commenti vanno scritti in inglese. Tale scelta è motivata dalle seguenti considerazioni:

- la maggioranza delle parole chiave presenti nei linguaggi di programmazione sono in lingua inglese, aggiungere commenti nella medesima lingua risulta più spontaneo e naturale.
- dare la possibilità di leggere il codice sorgente in maniera agevole anche a chi non è madrelingua italiano;
- alcuni strumenti di sviluppo in grado di individuare errori ortografici nel codice supportano solamente la lingua inglese;

2.2.6.5 Convenzioni per la documentazione

Le convenzioni impiegate nella generazione della documentazione sono le seguenti:

- **TODO:** utilizzato per indicare codice di cui verrà sicuramente effettuato un refactoring$_G$ in futuro. La stringa maiuscola TODO, inserita all'inizio dei commenti, viene fatta seguire dal carattere **:** (due punti); risulta una soluzione da praticare solamente nei casi in cui vi sia l'impossibilità temporanea di trovare una soluzione più appropriata al problema rilevato;
- **Intestazione:** i file sorgenti consegnati, ad esclusione di quelli di configurazione, devono iniziare con la seguente intestazione, adeguatamente contenuta in un commento di blocco:

```
1 /**
2 * Project: Predire in Grafana
3 * File: NameFile
4 * Author: Name Surname
5 * Created: YYYY-MM-DD
6 * Version: versionNumber
7 *-------------------------------------------------------------------------------------------
8 * Copyright 2020 ProApes Group.
9 * Licensed under the MIT License. See License.txt in the project root for license information.
10 *-------------------------------------------------------------------------------------------
11 * Changelog:
12 * #entry || Description || Author || Date
13 */
14
15
```

Figura 27: Intestazione dei file sorgenti

La versione del file nell'intestazione deve rispettare la formulazione:

[X].[Y].[Z] - [A].[B]

come espresso in §3.2.4.1 e 3.2.4.2. I numeri di versione del tipo X.0.0, dalla 1.0.0, sono considerate versioni stabili e quindi versioni da testare per saggiarne la qualità.

2.2.6.6 Convenzioni di nomenclatura dei file

- Classi, metodi, variabili devono avere un nome univoco ed esplicativo per evitare quanto più possibile incomprensioni;
- Le costanti vanno scritte usando solo maiuscole;
- Per i nomi di cartelle, file e classi la prima lettera deve essere posta in maiuscolo, col rispetto della notazione di tipo CamelCase nel caso di nomi composti;
- Per i nomi di variabili e metodi l'iniziale deve essere minuscola. Nel caso di nomi composti da più parole ciascuna prima lettera deve essere posta in maiuscolo. Nel caso di sigle o acronimi, devono essere trattate come parole.

Di seguito vengono presentati alcuni esempi esplicativi:

```
const MY_CONST
class MyClass
Folder
File.js
let myVariable
myMethod(){...}
```

2.2.6.7 Stile di codifica adottato

Nonostante i linguaggi di codifica adottati siano molteplici e di diversa natura (sia per la programmazione di back-end$_{\mathbf{G}}$ che per quella di font-end$_{\mathbf{G}}$, per esempio), il gruppo ha individuato alcune regole che possono essere applicate a tutti quanti indistintamente.
Ciascun membro del gruppo nella fase di generazione del codice è quindi tenuto a rispettare le seguenti norme:

Indentazioni

- I blocchi innestati devono seguire una corretta indentazione, usando per ciascun livello quattro (4) spazi;
- Per andare a capo di una linea di codice usare otto (8) spazi;
- È vietato l'utilizzo di tabulazioni, che devono essere necessariamente sostituite da spazi. Per garantire il rispetto di tale regola viene consigliato di configurare adeguatamente il proprio editor o IDE$_{\mathbf{G}}$.

Eccezioni: Queste regole non sono da applicarsi ai commenti.

Parentesi

- I blocchi di codice vanno racchiusi tra parentesi graffe;
- Le parentesi di delimitazione dei costrutti vanno inserite in linea e non al di sotto di essi;
- Fra il nome della classe/metodo e la parentesi graffa di apertura va inserita una singola spaziatura.

Brevità dei metodi

In generale è sempre desiderabile se possibile, definire dei metodi brevi (concentrati sullo scopo), dove far corrispondere ciascuna riga ad una singola istruzione.
A volte definire metodi "lunghi" risulta tuttavia indispensabile, quindi questa non è una norma obbligatoria ma consigliata. Se il corpo del metodo dovesse superare le quaranta (40) righe di codice sarebbe necessario suddividerlo, senza però minare la struttura del programma.

Lunghezza delle righe di codice

Una riga di codice deve essere lunga al massimo 140 caratteri; in caso si superi questo valore la riga va spezzata in due.

Codice esterno

Se si ha a che fare con codice proveniente da fonti esterne è consigliato e preferibile rispettare lo stile e le norme di codifica degli autori di quel codice.

Ricorsione

Potendo portare ad una maggiore occupazione di memoria rispetto alle soluzioni iterative, l'uso della ricorsione va evitato il più possibile. Nel caso non fosse realizzabile, il suo impiego va giustificato adeguatamente attraverso commenti.

2.2.6.8 *ECMAScript 6 (TypeScript)*

Il gruppo ha deciso di adottare *TypeScript*$_{\mathbf{G}}$ come linguaggio di programmazione principale e *ECMAScript*$_{\mathbf{G}}$ come specifica tecnica, in quanto suggerito dalla guida sviluppatori di *Grafana*$_{\mathbf{G}}$. Come stile di codifica si è scelto di adottare le convenzioni proposte dalla *Airbnb JavaScript Style Guide* - riconducibili anche al linguaggio *TypeScript* utilizzato dal gruppo.

`https://github.com/airbnb/javascript`

Una particolare attenzione viene posta sulle seguenti linee guida, in quanto ritenute particolarmente utili per la realizzazione del prodotto *Predire in Grafana* (le seguenti indicazioni dei paragrafi fanno riferimento alla guida sopracitata):

- utilizzare `const` e `let`, evitando di usare `var` per la dichiarazione delle variabili (§2.1, §2.2):
  - `const`: utilizzato per dichiarare costanti, in questo modo si ha la certezza che non possano essere riassegnate, e si evitano possibili bug;
  - `let`: utilizzato per dichiarare variabili *block-scoped*, che quindi, al contrario di quelle dichiarate `var`, non sono visibili ad altri blocchi della funzione;
- utilizzare `Array.push()` invece di assegnare direttamente gli oggetti in un array (§4.2). Tale metodo si basa sulla lunghezza dell'array per inserire il nuovo elemento, e quando questa è zero (0) o non definita, la inizializza;
- utilizzare l'operatore `...` per copiare un array (§4.3);
- preferire la notazione "a freccia" (`=>`) per le funzioni anonime (§8.1); essa definisce una funzione con una sintassi più concisa ed esegue implicitamente nel contesto `this`, che normalmente è quello che si desidera;
- utilizzare `import/export` per i moduli (§10.1), evitando le altre soluzioni non standard deprecate.

Commenti al codice

I commenti devono essere in lingua inglese. Inseriti esclusivamente ove necessari a migliorare la leggibilità del codice, possono essere di diversi tipi a seconda delle necessità:

- **linea singola**: per descrivere la singola istruzione che li precede:

```
// single-line-comment
```

- **linea multipla**: per descrivere il funzionamento di blocchi più lunghi di codice:

```
/** multi-line-comment */
```

2.2.6.9 *ReactJS*

Vista l'intenzione della piattaforma *Grafana*$_{\mathbf{G}}$ di supportare esclusivamente il framework *ReactJS*$_{\mathbf{G}}$ a partire dalla versione attualmente attiva, si è scelto di utilizzare questo framework anche per la realizzazione della parte front-end del *plug-in*$_{\mathbf{G}}$. In particolare, saranno tenute in considerazione le linee guide descritte nella *Airbnb React/JSX Style Guide*.

`https://github.com/airbnb/javascript/tree/master/react`

2.2.6.10 *HTML e CSS*

Per le parti di codice riguardanti la visualizzazione dell'interfaccia utente, il framework *ReactJS* viene utilizzato in combinazione con i linguaggi *HTML* e *CSS*. Il gruppo ha scelto per questi ultimi di fare riferimento alle linee guida suggerite nella *Google HTML/CSS Style Guide*.

`https://google.github.io/styleguide/htmlcssguide.html`

In particolare, si evidenziano le norme riguardanti la validazione *HTML5*$_{\mathbf{G}}$ (§3.1.1, §3.1.2), la semantica (§3.1.3) e la formattazione (§3.2), al fine di migliorare la leggibilità del codice.

2.2.6.11 Metriche di prodotto

Per la valutazione dei prodotti del processo di codifica, il gruppo adotterà nelle prossime fasi delle opportune metriche di qualità; non è stato ritenuto opportuno applicarne alcuna al PoC, in quanto prodotto software ancora immaturo e non significativo in termini di qualità finale.

- **MPDS01** Copertura requisiti obbligatori (per una trattazione più estesa, consultare §D.2.3);
- **MPDS02** Copertura requisiti accettati (per una trattazione più estesa, consultare §D.2.4);
- **MPDS03** Numero di *bug* (per una trattazione più estesa, consultare §D.2.5);
- **MPDS04** Numero di *code smell* (per una trattazione più estesa, consultare §D.2.6);
- **MPDS05** *Technical debt* (per una trattazione più estesa, consultare §D.2.7);
- **MPDS06** *Remediation effort* (per una trattazione più estesa, consultare §D.2.8);
- **MPDS07** Complessità ciclomatica (per una trattazione più estesa, consultare §D.2.9);
- **MPD04** Complessità cognitiva (per una trattazione più estesa, consultare §D.2.10);
- **MPDS08** Successo dei test (per una trattazione più estesa, consultare §D.2.11);
- **MPDS09** *Line coverage* (per una trattazione più estesa, consultare §D.2.12);
- **MPDS11** *Branch coverage* (per una trattazione più estesa, consultare §D.2.13);
- **MPDS12** *Code coverage* (per una trattazione più estesa, consultare §D.2.14);
- **MPDS13** Densità di duplicazione (per una trattazione più estesa, consultare §D.2.15);
- **MPDS14** Rapporto fra linee di commento e linee di codice (per una trattazione più estesa, consultare §D.2.16);
- **MPDS15** Numero di nuove righe (per una trattazione più estesa, consultare §D.2.17).

2.2.7 Strumenti

Di seguito vengono presentati gli strumenti che il gruppo ha deciso di adottare all'interno del progetto durante il processo di Sviluppo.

Visual Studio Code

Il gruppo ha deciso di impiegare questo IDE$_{\mathbf{G}}$ per la stesura di tutto il codice, sia per il modulo interno che quello esterno.

https://code.visualstudio.com/

I fattori principali che ne hanno comportato la scelta riguardano:

- la sua elevata popolarità, garantita anche dai frequenti aggiornamenti rilasciati dal colosso Microsoft e da un'affezionata community;
- la sua integrazione nativa con *TypeScript*$_{\boldsymbol{G}}$ e *Git*$_{\boldsymbol{G}}$;
- le performance nell'indicizzazione dei file e i potenti strumenti di ricerca;
- la sua ricca rete di plug-in$_{\mathbf{G}}$ (sia ufficiali, sia sviluppati dalla community);
- il fatto che alcuni membri del team ne avessero già conoscenza, anche se superficialmente.

InfluxDB

Database specializzato nella gestione di grandi quantità di dati. Esso viene interrogato da *Grafana*$_{\boldsymbol{G}}$ attraverso la presentazione di dashboard$_{\mathbf{G}}$ di grafici e notificando allarmi.

Draw.io

Per la generazione di diagrammi UML$_{\mathbf{G}}$ il gruppo ha deciso di utilizzare *Draw.io*$_{\boldsymbol{G}}$ in quanto offre molte agevolazioni che permettono una produzione veloce dei diagrammi e risulta uno strumento di semplice apprendimento e uso.

https://www.draw.io

3 Processi di Supporto

3.1 Documentazione

3.1.1 Scopo

Ogni processo$_{\mathbf{G}}$ e attività$_{\mathbf{G}}$ significativi per lo sviluppo del progetto saranno documentati. Nella presente sezione verranno descritte le regole e gli standard$_{\mathbf{G}}$ che disciplineranno il processo$_{\mathbf{G}}$ di Documentazione durante l'intero ciclo di vita$_{\mathbf{G}}$ del software, permettendo di ottenere prodotti documentali coerenti e validi dal punto di vista tipografico e formale.

3.1.2 Descrizione

Di seguito vengono presentate le decisioni e le norme prescelte per la stesura, verifica e approvazione della documentazione ufficiale interna ed esterna; ogni membro del gruppo *ProApes* è tenuto ad attenervisi.
Il processo di Documentazione è così strutturato:

- suddivisione dei documenti;
- struttura dei documenti;
- produzione;
- mantenimento.

3.1.3 Aspettative

Le attese del team per questo processo sono:

- individuare una struttura comune per tutti i prodotti del processo, nell'arco del ciclo di vita del software;
- raccogliere tutte le norme comuni per la stesura dei documenti ufficiali.

3.1.4 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$_{\mathbf{G}}$ remoto (vedi sezione §3.2.5.1);
- **Strutturazione:** il documento viene fornito di:
  - un registro delle modifiche$_{\mathbf{G}}$;
  - un indice dei contenuti;
  - un indice delle figure e un indice delle tabelle (se necessari).
- **Stesura:** i membri del gruppo redigono il corpo del documento adottando un metodo incrementale;
- **Revisione:** ogni 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 rilasciato.

3.1.5 Classificazione dei documenti

I documenti prodotti durante questo processo saranno di due tipi:

- **Formali:** riportano le norme che regolano l'operato del gruppo e gli esiti delle attività da esso portate avanti nel corso del ciclo di vita del software. Le caratteristiche di un documento formale sono:
  - storicizzazione delle versioni$_{\mathbf{G}}$ 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 presenta più versioni, si considera come corrente sempre la più recente tra quelle approvate dal *Responsabile di Progetto*. I documenti formali possono essere:
  - **Interni:** riguardanti le dinamiche interne del gruppo, di marginale interesse per committenti$_{\mathbf{G}}$ e proponente$_{\mathbf{G}}$;
  - **Esterni:** d'interesse per committenti e proponente, vengono loro consegnati nell'ultima versione approvata.

  Sono documenti formali:
  - ***Norme di Progetto:*** contenente le norme e le regole, stabilite dei membri del gruppo, alle quali ci si dovrà attenere durante l'intera durata del lavoro di progetto. Documento interno;
  - ***Glossario:*** elenco ordinato di tutti i termini usati nella documentazione che il gruppo ritiene necessitino di una definizione esplicita; documento esterno;
  - ***Studio di Fattibilità:*** analisi critica di tutti i capitolati a disposizione, con valutazione degli aspetti positivi e negativi di ognuno. Contenente anche l'indicazione del capitolato eletto dal gruppo, è un documento interno;
  - ***Piano di Progetto:*** espone la pianificazione di tutte le attività di progetto previste dal gruppo *ProApes*, presentando una previsione dell'impegno orario dei singoli membri, un preventivo delle spese e tutti i consuntivi di periodo; documento esterno;
  - ***Piano di Qualifica:*** espone e descrive i criteri di valutazione della qualità adottati dal gruppo; documento esterno;
  - ***Analisi dei Requisiti:*** presenta tutti i requisiti e le caratteristiche che il prodotto finale dovrà avere; documento esterno.
- **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 del team; conterranno un ordine del giorno, gli argomenti affrontati e le decisioni prese;
  - **Esterni:** rapporti degli incontri del gruppo coi committenti e/o col proponente; la trattazione avviene per punti ciascuno accompagnato dal riassunto di quanto emerso.

  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.

3.1.5.1 Nomenclatura dei documenti

- La denominazione dei documenti formali seguirà lo schema:

  [NomeDocumento]_v[X].[Y].[Z] - [A].[B]

  dove

  - **[NomeDocumento]**, corrispondente al nome ufficiale del documento. Non sono permessi spazi all'interno del nome e i caratteri devono essere tutti minuscoli ad eccezione delle iniziali (convenzione *CamelCase*);
  - **v[X].[Y].[Z] - [A].[B]** in cui **v** sta per "versione" e **X**, **Y**, **Z**, **A** e **B** rispettano quanto indicato in §3.2.4.1 e §3.2.4.2.

- La denominazione dei documenti informali (i verbali) seguirà lo schema:

  V[I/E]_[YYYY]-[MM]-[DD]

  dove

  - **Verbale:** indica il tipo di documento;
  - **[I/E]:** indica se il verbale rendicontato è interno o esterno;
  - **[YYYY]-[MM]-[DD]:** indica la data dell'incontro:
    - **[YYYY]** corrisponde all'anno;
    - **[MM]** corrisponde al mese;
    - **[DD]** corrisponde al giorno.

3.1.6 Directory di un documento

Ogni directory$_{\mathbf{G}}$ prende il nome dal documento che contiene secondo la forma:

NomeDocumento

e viene posizionata, a seconda del tipo di documento, nella directory **DocumentiInterni** o **DocumentiEsterni**.
Dentro la directory si trovano i file specifici del documento: quelli dedicati al frontespizio e quelli dedicati al contenuto; questi ultimi sono raggruppati in una cartella **content**. I singoli file sono analizzati in §3.1.7.2.

3.1.7 Struttura dei documenti

3.1.7.1 Template

Il gruppo ha deciso di adottare il linguaggio LaTeX per la stesura dei documenti; è 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 rende più efficiente l'applicazione di nuove norme o di modifiche a norme già esistenti a tutti i documenti redatti fino a quel momento.

3.1.7.2 Struttura documenti formali

Un file "NomeDocumento.tex" raccoglie, grazie a comandi di input, le sezioni che compongono il documento, oltre ai file per la configurazione e lo stile. I file di input sono:

- **packages.tex:** contenente i pacchetti necessari alla compilazione;
- **command.tex:** contiene i comandi LaTeX creati dal team;
- **style.tex:** contenente le direttive per l'impaginazione dei documenti;

- **history.tex:** contiene il registro delle modifiche;
- **firstpage.tex:** contiene il frontespizio coi dati specifici del documento;
- **Introduzione.tex:** paragrafo d'introduzione;
- **TitoloSezione.tex:** uno per ogni sezione del contenuto, in numero variabile a seconda del tipo di documento.

Un documento resta in formato LaTeX finché non viene approvato dal *Responsabile di Progetto*, dopo di che viene convertito in formato *PDF*.

Frontespizio

La prima pagina di ogni documento sarà composta dagli elementi sotto esposti:

- **Logo del gruppo** posto in alto al centro;
- **Indirizzo e-mail del gruppo** posto subito sotto il logo, al centro;
- **Nome del gruppo** al centro della pagina;
- **Informazioni sul documento**, che includono:
  - **Versione:** indica la versione corrente, approvata dal *Responsabile di Progetto*;
  - **Data di approvazione:** indica la data in cui si è conclusa la stesura del documento;
  - **Responsabile:** indica il nome del *Responsabile di Progetto* che ha approvato il documento;
  - **Redattori:** i membri di *ProApes* che si sono occupati della stesura del documento;
  - **Verificatori:** i membri di *ProApes* che si sono occupati della verifica del documento;
  - **Stato di approvazione:** indica se il documento è stato approvato dal *Responsabile di Progetto*; deve essere sempre *Approvato* per un documento formale;
  - **Uso:** indica se il documento è destinato ad uso *interno* o *esterno*;
  - **Lista di distribuzione:** indica i destinatari del documento;
- **Sommario**, posto in fondo alla pagina, contiene una breve descrizione del contenuto del documento.

Diario delle modifiche

Ogni documento presenta un registro delle modifiche, sotto forma di tabella, che tiene traccia di tutte le modifiche significative apportate al documento durante le fasi del suo ciclo di vita.
Ciascuna voce della tabella riporta:

- una sintetica descrizione della modifica apportata;
- il nome dell'autore della modifica;
- il ruolo ricoperto dall'autore quando ha eseguito la modifica;
- la data in cui è stata apportata tale modifica;
- la versione del documento dopo la modifica.

I riferimenti alle sezioni, di norma numerici, saranno ottenuti mediante un apposito comando LaTeX; l'unica eccezione si avrà nel caso di riferimenti a sezioni inserite in una prima versione del documento ma poi rimosse: qui il comando non sarà più adatto. Per mantenere tuttavia la consistenza del Diario delle Modifiche, il gruppo ha deciso di indicare a parole (senza l'uso di comandi LaTeX), i titoli delle sezioni eliminate ma di cui ancora esistono riferimenti in tabella.

Indici

L'indice delle sezioni fornisce al fruitore una visione complessiva della struttura del documento, permette di orientarsi tra i contenuti e di individuare la posizione delle varie parti.
Ogni documento presenta un indice dei contenuti, subito dopo il diario delle modifiche; dove necessario, sono presenti anche un indice delle illustrazioni e uno delle tabelle presenti nel documento.

Corpo del documento

Ogni pagina nel corpo del documento deve rispettare delle specifiche regole:

- in alto a sinistra si trova una miniatura a colori del logo del gruppo;
- in alto a destra viene indicato il titolo del documento;
- sotto i due elementi appena elencati si trova una linea nera continua, che li separa dal testo della pagina;
- il contenuto della pagina si trova sotto la linea;
- una linea nera continua separa il contenuto della pagina dal piè di pagina;
- al centro del piè di pagina, sotto la linea, è indicato il numero della pagina corrente e il numero totale di pagine del documento, nella forma "[numeroPagina] di [numeroTotale]".

3.1.7.3 Struttura dei verbali di riunione

Ai verbali, sia interni che esterni, si applicano le medesime norme strutturali degli altri documenti, ad eccezione del fatto che, essendo **informali**, non sono soggetti a versionamento. Non si è inoltre ritenuto necessario andare a evidenziare eventuali termini di glossario.
Il contenuto di un verbale viene così presentato:

- **Luogo:** sede in cui si è tenuto l'incontro;
- **Strumento:** se l'incontro non si è potuto svolgere in presenza indica la piattaforma impiegata (e.g: *Discord, Skype* ecc.). In questo caso il *Luogo* sarà contraddistinto da un "-" (trattino);
- **Data:** riporta la data della riunione;
- **Ora di inizio:** orario d'inizio della seduta;
- **Ora di fine:** orario di fine della seduta;
- **Partecipanti:** elenco dei presenti all'incontro;
- **Ordine del giorno:** elenco degli argomenti affrontati durante l'incontro;
- **Resoconto:** esito sintetico della discussione dei singoli punti inseriti nell'ordine del giorno.
- **Tracciamento delle decisioni:** riassunto in forma tabellare delle decisioni prese dal gruppo durante un incontro interno o esterno; ad ogni decisione viene assegnato un codice così strutturato:

  V[I/E]_[YYYY]-[MM]-[DD].[X]

  dove la prima parte è analoga al nome del verbale, mentre **X** indica il numero progressivo della decisione all'interno di quel verbale, a partire da 1.

3.1.8 Norme tipografiche

Per evitare incongruenze tra i vari file, di seguito vengono esposte le norme riguardanti l'ortografia, la tipografia e l'adozione di uno stile uniforme in tutti i documenti.

3.1.8.1 Convenzioni di denominazione

I nomi di file e cartelle specifici di un singolo documento iniziano con la lettera maiuscola e, quando composti da più parole, le presentano attaccate ma distinguibili dall'iniziale maiuscola. Per file e cartelle legati alla struttura del documento (come la cartelle *img*, *content* o i file *history.tex*, *firstpage.tex*) i nomi non contengono caratteri maiuscoli.
Esempi **corretti**:

- NormeDiProgetto;
- VerbaleInterno;
- commands.tex.

Esempi **errati**:

- Analisi_dei_requisiti (contiene caratteri separatori);
- processidisupporto.tex (non distingue le parole mediante le iniziali maiuscole);
- pianoDiProgetto (inizia con la lettera minuscola);

3.1.8.2 Stile del testo

Gli stili di testo adottati nei documenti sono:

- **grassetto:** per i titoli, sottotitoli, il nome degli oggetti negli elenchi puntati (seguiti da una descrizione), termini ritenuti importanti dal *Redattore*;
- **corsivo:** per i nomi propri (precisamente i membri del gruppo, il proponente, i committenti), i nomi dei ruoli, il nome del progetto, i nomi dei documenti, i nomi delle tecnologie[5], degli strumenti, per formule matematiche, termini specifici e citazioni;
- **monospace:** per gli snippet$_{\mathbf{G}}$ di codice;
- **maiuscolo:** per gli acronimi, per le iniziali dei nomi dei documenti (secondo la convenzione *CamelCase*$_{\mathbf{G}}$) dei processi e delle attività.

3.1.8.3 Termini di glossario

I termini il cui significato potrebbe essere ambiguo vengono contrassegnati con una **G** a pedice (e.g: glossario$_{\mathbf{G}}$) alla loro prima occorrenza nella sezione d'interesse. Se la voce viene ripetuta più volte nella stessa sezione, è sufficiente contrassegnarla solo la prima volta che compare. Se un termine viene usato e subito spiegato, non è necessario rimandare al glossario, poiché risulterebbe ridondante.
Tutti i termini da glossario sono riportati in ordine alfabetico in un documento esterno omonimo, il *Glossario*.

3.1.8.4 Elementi testuali

Durante la stesura della documentazione i *Redattori* incaricati devono attenersi alle regole stilistiche sotto esposte:

[5]Il termine *tecnologie* è da intendersi non solo in senso stretto ma considerando anche ciò che appare nello *Studio di Fattibilità 1.0.0-1.10*

Elenchi puntati

Il simbolo per scandire un elenco puntato è il • (pallino); al livello successivo di annidamento, il simbolo usato è – (doppio trattino); al terzo livello è * (asterisco).
Se si tratta di un elenco numerato, i tre livelli di enumerazione sono ordinati coi numeri arabi seguiti da un punto fermo (**1.**), con le lettere dell'alfabeto latino tra parentesi tonde (**(a)**) e dai numeri romani in caratteri minuscoli seguiti da punto fermo (**i.**) rispettivamente.
Ogni voce di un elenco inizia con la lettera minuscola (tranne per i nomi propri), e termina con **;** (punto e virgola), a meno che non si tratti dell'ultima voce, la quale termina con **.** (punto fermo).
Le voci che corrispondono allo schema *NomeElemento-descrizione* presentano il primo termine della coppia in grassetto, seguito dal secondo elemento in formato normale.

Formati di dato

Per l'indicazione delle date e degli orari si segue quanto sancito dallo standard ISO $8601_{\mathbf{G}}$.
Le date vengono indicate usando il formato:

[YYYY]-[MM]-[DD]

dove

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

Per gli orari viene seguita l'usuale convenzione:

[HH]:[MM]

dove

- **HH** rappresenta le ore;
- **MM** rappresenta i minuti.

Nel caso in cui le ore e/o i minuti dovessero essere di una singola cifra si deve anteporre uno zero ad essa.

Sigle

Tutte le sigle sono indicate con le iniziali di ogni parola maiuscola.
Le sigle usate nei documenti del progetto sono:

- relative ai nomi dei documenti, per le quali vale che le preposizioni, le congiunzioni e gli articoli che compongono il nome abbiano l'iniziale minuscola:
  - **Analisi dei Requisiti:** AdR;
  - **Piano di Progetto:** PdP;
  - **Piano di Qualifica:** PdQ;
  - **Glossario:** G;
  - **Studio di Fattibilità:** SdF;
  - **Norme di Progetto:** NdP;
  - **Verbali Interni:** VI;
  - **Verbali Esterni:** VE;
  - **Manuale Sviluppatore:** MS;

  - **Manuale Utente:** MU.

- relative alle revisioni di progetto$_{\mathbf{G}}$ previste dai committenti:
  - **Revisione dei Requisiti:** RR;
  - **Revisione di Progettazione:** RP;
  - **Revisione di Qualifica:** RQ;
  - **Revisione di Accettazione:** RA.

- relative ai ruoli di progetto:
  - **Responsabile di progetto:** RE;
  - **Amministratore:** AM;
  - **Analista:** AN;
  - **Progettista:** PT;
  - **Programmatore:** PR;
  - **Verificatore:** VE.

Note a piè di pagina

Ogni nota a piè pagina dovrà iniziare con un riferimento numerico in apice seguito dalla prima lettera maiuscola della prima parola del paragrafo.

Nomi

Per i nomi usati frequentemente all'interno dei documenti sono stati implementati dei comandi LaTeX personalizzati:

- **nomi propri di persona:** presentati nella forma del nome seguito dal cognome; per i nomi dei membri del gruppo *ProApes* è stato creato un comando LaTeX personalizzato: `\nome` (nome della persona), che ne garantisce la visualizzazione nel formato corretto;
- **nome del gruppo:** *ProApes*; scritto con le prime due lettere maiuscole; il comando LaTeX creato è `\authorName`;
- **nome del proponente:** ci si riferirà al proponente con "*Zucchetti S.p.A.*" oppure con "proponente"; nel primo caso `\proposerName` è il comando da impiegare;
- **nomi dei committenti:** ci si riferirà ai committenti con "*Prof. Tullio Vardanega* e *Prof. Riccardo Cardin*" oppure con "committenti"; grazie ad una coppia di comandi LaTeX personalizzati: `\commitName` e `\commitNameC`" è possibile visualizzare direttamente i nomi dei committenti;
- **Nome del progetto**: come nei punti sopra esposti anche il nome del progetto "Predire in Grafana" vede l'impiego di un comando LaTeX personalizzato: `\project`;
- **Nome dei documenti:** i nomi dei documenti di progetto andranno formattati utilizzando la prima lettera maiuscola per ogni parola che non sia di preposizione (e.g: *Norme di Progetto*); a tal fine viene messo a disposizione un comando apposito che permette la comparsa della versione affianco al nome del documento: `\NdP` da luogo a *Norme di Progetto 4.0.0-1.10*.

3.1.8.5 Elementi grafici

Questa sezione definisce le norme per l'uso di elementi grafici quali immagini, tabelle e diagrammi.

Immagini

Le figure presenti nei documenti sono tutte centrate rispetto al testo e accompagnate da un'opportuna didascalia.

Grafici UML

I grafici in linguaggio UML$_{\mathbf{G}}$, usati per la modellazione dei casi d'uso$_{\mathbf{G}}$ e per i diagrammi della progettazione$_{\mathbf{G}}$, sono inseriti come immagini.

Tabelle

Ogni tabella è contrassegnata da una didascalia descrittiva del contenuto, posta sotto di essa e centrata rispetto alla pagina. Nella didascalia di ogni tabella viene indicato l'identificativo

Tabella [X]

dove **[X]** indica il numero assoluto della tabella all'interno del documento; segue poi il testo della didascalia.
A questa prassi fanno eccezione le tabelle del registro delle modifiche, che non hanno didascalia né numero.

3.1.9 Metriche

Per la valutazione dei prodotti del processo di documentazione, il gruppo ha deciso di adottare le seguenti metriche$_{\mathbf{G}}$ di qualità di prodotto:

- **MPDD01** Indice Gulpease (per una trattazione più estesa, consultare §D.2.3);
- **MPDD02** Errori ortografici (per una trattazione più estesa, consultare §D.2.4).

3.1.10 Strumenti

Gli strumenti usati per la stesura dei documenti sono il linguaggio LaTeX, l'editor *Texmaker* e *Draw.io*.

LaTeX

Per la stesura dei documenti, il gruppo *ProApes* ha scelto LaTeX, un linguaggio compilato basato sul programma di composizione tipografica TeX, per ottenere documenti coerenti, ordinati, templatizzati e stesi in modo collaborativo.

Texmaker

L'editor scelto per la stesura dei documenti è *Texmaker*, adottato da tutti i membri del gruppo. *Texmaker* fornisce strumenti di compilazione del testo composto e visualizzazione del *PDF* risultante fruibili in modo agevole.

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

Draw.io

Per la produzione di grafici UML è stato scelto il programma *Draw.io*.

https://www.draw.io

3.1.11 Verifica ortografica

Per la verifica ortografica dei documenti scritti in LaTeX si utilizzerà lo strumento di correzione automatica presente nell'editor *Texmaker*$_{\boldsymbol{G}}$ oppure lo strumento *Aspell* (versione $> 0.60.5$). Il primo fornisce avvisi di errore istantaneamente; il secondo deve essere utilizzato da terminale lanciando il comando direttamente sul file in formato tex:

```
aspell -mode=tex -lang=it check NomeFile.tex
```

3.2 Gestione della configurazione

3.2.1 Scopo

Lo scopo del processo è gestire in modo ordinato e sistematico la produzione di documenti e codice. Un elemento soggetto a configurazione ha una collocazione, una denominazione e uno stato definiti; per ogni oggetto configurato sono garantiti la modifica normata e il versionamento.

3.2.2 Descrizione

Il processo di Gestione della configurazione raggruppa e organizza tutti gli strumenti necessari alla configurazione degli strumenti adoperati per la produzione di documenti, codice e diagrammi, ma anche quelli necessari al versionamento e al coordinamento del gruppo.

3.2.3 Aspettative

Le attese, per l'implementazione di questo processo, sono:

- rendere sistematica la produzione di codice e documentazione;
- unificare ed uniformare lo stato degli strumenti usati durante lo svolgimento di tutto il progetto;
- classificare i prodotti dei vari processi implementati.

3.2.4 Versionamento

3.2.4.1 Codice di versione per documenti e software

La storia di una componente del prodotto delle attività di progetto deve essere sempre ricostruibile attraverso le sue versioni. Il codice di una versione è nel formato

[X].[Y].[Z]

dove

- **X** indica un rilascio pubblico e corrisponde ad una versione del documento approvata dal *Responsabile di Progetto*; la numerazione inizia da 0.
- **Y** indica una revisione complessiva del prodotto per verificarne la coesione e consistenza a fronte di modifiche circoscritte apportate a parti diverse; concettualmente potrebbe essere confrontato alla funzione dei test di integrazione nel software; la numerazione inizia da 0 e riparte da questo valore ad ogni incremento di **X**;
- **Z** viene incrementato ad ogni modifica circoscritta e relativa verifica; concettualmente potrebbe essere confrontato alla funzione dei test di unità nel software; la numerazione inizia da 0 e riparte da tale valore ad ogni incremento di **X** o **Y**.

3.2.4.2 Codice di versione unitario

Entrando nel vivo di Progettazione e Codifica, il gruppo ha avvertito la necessità di dare unità al proprio prodotto finale, composto da documenti e componenti software.
Per mantenere la natura composita del prodotto ma al contempo renderlo un *unicum* il gruppo ha deciso di conservare la versione della singola componente (descritta in §3.2.4.1) e di affiancarvi una coppia di indici secondo la sintassi

-[A].[B]

dove

- **A** indica una versione completa e funzionante del prodotto: le componenti software implementano tutti i requisiti obbligatori, i test sono stati tutti superati, le metriche di qualità soddisfatte, e sono presenti versioni approvate di tutti i documenti richiesti per il prodotto finale; viene incrementato quando il prodotto software non è più retrocompatibile; la numerazione parte da 0 e non viene annullata;
- **B** cresce al raggiungimento degli obiettivi degli incrementi pianificati nel *Piano di Progetto 4.0.0-1.10*: poiché tali incrementi presentano obiettivi sia documentali che software, la crescita di questo indice rappresenta un incremento nello sviluppo totale del prodotto; la numerazione parte da 0 e non viene annullata.

3.2.4.3 Tecnologie adottate

Per la gestione delle versioni è stato scelto il sistema di versionamento distribuito *Git*$_{\mathbf{G}}$, con due repository remoti posizionati in *GitHub*$_{\mathbf{G}}$.

3.2.5 Struttura dei repository

Il gruppo *ProApes* ha deciso di creare, per il progetto, un repository suddiviso in tre *submodules*, ovvero che al suo interno contiene il collegamento ai tre repository di codice e documentazione:

- **ModuloInterno** per il versionamento del codice relativo al plug-in di *Grafana*$_{\mathbf{G}}$;
- **ModuloEsterno** per il versionamento del codice relativo al modulo di addestramento;
- **Documentation** per il versionamento dei documenti redatti.

Si è optato per la separazione del codice dai documenti per tenere disgiunte le sezioni che riguardano la parte d'implementazione da quelle che raccolgono la documentazione ad essa relativa.
Tuttavia è stato possibile collegare i due moduli in nell'unico repository **ProApes** contenente tutta la produzione del gruppo:

`https://github.com/Kero2375/ProApes`

Esiste una versione del repository **remota** e una **locale**, con la medesima struttura:

- **remoto**: presente su *GitHub*, contiene i prodotti condivisi dai vari membri del gruppo;
- **locale**: lanciando il comando da terminale

  ```
  git clone [URLRepository]
  ```

  ogni membro del gruppo ottiene una copia del repository remoto in locale: sarà la cartella in cui ognuno produce codice o documenti.

3.2.5.1 Repository per la documentazione

Il repository **ProApesDocumentation** è così organizzato:

- **Template**: cartella contenente i file del template in linguaggio LaTeX, usati per assemblare tutti i documenti;
- **R***: cartella contenente a sua volta le sottocartelle dei file sorgenti (nel formato *.tex*) dei singoli documenti necessari alla *Revisione* corrente, distinti tra interni ed esterni; lo schema verrà adottato con le cartelle **RR**, **RP**, **RQ**, **RA** rispettivamente per la *Revisione dei Requisiti*, *Revisione di Progettazione*, *Revisione di Qualifica* e *Revisione di Accettazione*.

La divisione dei file in base alla revisione di appartenenza favorisce una classificazione ordinata dei prodotti della documentazione ed una loro semplice tracciabilità.

Tipi di file

Nelle singole cartelle dei documenti sono inclusi:

- file *README.md* con le informazioni generali sullo scopo del repository, sull'identità del gruppo e sulle finalità del prodotto versionato;
- file *.tex* per i sorgenti in LaTeX;
- file *.pdf* per i documenti oggetto di consegna;
- immagini da inserire nei documenti;

Il file *.gitignore* contenente un riferimento a tutti i file non versionati; tale file è situato nel livello più esterno del repository.

3.2.5.2 Repository per il codice

Il repository **ProApesCode** è così organizzato:

- **.github/workflows**: cartella contenente i file per la configurazione della *Continuous Integration*, implementata mediante l'uso di *GitHub Actions*;
- **training-module**: cartella contenente file di configurazione e sorgenti per il modulo esterno di addestramento$_{\mathbf{G}}$;
- **prediction module**: cartella contenente file di configurazione e sorgenti per il modulo interno$_{\mathbf{G}}$predittivo;
- **DataText**: file *.txt* contenente alcuni dati fittizi per simulare il funzionamento del modulo esterno di addestramento;

Tipi di file

Nelle singole cartelle dei moduli sono inclusi:

- file *README.md* con le istruzioni per la configurazione dell'ambiente e il lancio del prodotto software, diverso per i due moduli;
- file *.json* con le informazioni che descrivono il *plug-in*$_{\mathbf{G}}$ e le sue dipendenze; fondamentali per il corretto riconoscimento da parte di *NodeJS*$_{\mathbf{G}}$ e di *Grafana*.
- file *.ts, .tsx, .js* contenenti codice *TypeScript*$_{\mathbf{G}}$ o *JavaScript*$_{\mathbf{G}}$ per la parte di logica e front-end dei moduli;
- file *.css* contenenti codice *CSS*$_{\mathbf{G}}$ per l'impostazione grafica dell'interfaccia utente.

Il file *.gitignore* contenente un riferimento a tutti i file non versionati.

3.2.6 Comandi base di GitHub

Il repository **ProApesDocumentation** è composto da diversi *branch*$_{\mathbf{G}}$:

- **master**, su cui vengono riportati i documenti nella loro versione definitiva ufficiale, approvata dal *Responsabile di Progetto*;
- **Develop**, ramo di sviluppo in cui vengono riportati i documenti terminati ma non ancora approvati dal *Responsabile di Progetto*;
- i rami dedicati ai singoli documenti, denominati secondo la convenzione *CamelCase*, uno per ogni documento da produrre, ad esclusione dei Verbali, per i quali sono adibiti solo un branch **VerbaliInterni** e uno **VerbaliEsterni**. Un documento resta sul ramo ad esso dedicato per tutta la sua stesura e verifica;

In modo analogo, il repository **ProApesCode** del codice è composto dai *branch* **master**, **develop**, e uno dedicato ad ogni nuova funzione che deve essere implementata all'interno dei plug-in.
Questa divisione è orientata ad uno sviluppo per *feature*$_{\mathbf{G}}$.
Per lavorare alla stesura di un documento nel proprio repository locale, ogni membro del gruppo deve:

1. posizionarsi nel proprio repository locale;
2. aprire un terminale;
3. spostarsi sul ramo dedicato al documento cui deve lavorare, mediante il comando `git checkout` seguito dal nome del branch desiderato;
4. eseguire il comando `git pull` per trasferire nel proprio repository locale eventuali modifiche apportate da altri membri del gruppo a quel ramo presenti in remoto;
5. svolgere la propria attività di scrittura nell'editor predisposto (in questo caso, *Texmaker*);
6. tornato sul terminale, eseguire il comando `git add` seguito dai nomi dei file che ha modificato;
7. eseguire il comando `git commit -m` seguito da una breve descrizione del compito svolto, inserita tra doppi apici;
8. eseguire il comando `git push` per pubblicare il proprio lavoro sul repository remoto e renderlo visibile a tutti i membri del gruppo.

In questo modo, ogni modifica eseguita in locale viene regolarmente riportata in remoto, garantendo una corrispondenza costante tra i prodotti elaborati dai singoli sulla propria macchina e quelli condivisi da tutti i membri.
Un documento resta sul proprio ramo durante tutta la stesura e verifica; terminata la verifica, il documento viene spostato sul ramo **Develop**, dove resterà fino a quando non sarà approvato dal *Responsabile di Progetto*. A quel punto, viene spostato sul ramo **master**.

3.2.7 Modifiche al repository

Tutti i membri del gruppo possono apportare modifiche ai file elaborati, tranne a quelli presenti nel ramo **master**. Per apportare cambiamenti permanenti ai file presenti su questo ramo è necessaria una *pull request*$_{\mathbf{G}}$, con conseguente approvazione di almeno un altro elemento del gruppo.
Per modifiche minime a codice e documenti è permessa la modifica autonoma da parte di qualunque membro del gruppo. Se necessario, questi deve essere in grado di giustificare il proprio intervento davanti al resto del team.
Per modifiche considerevoli a documenti già approvati o codice già verificato è necessario:

1. contattare il *Responsabile di Progetto* che ha approvato tale prodotto
2. esporgli la modifica che si intende fare e le motivazioni che la muovono;
3. se autorizzati, apportarla effettivamente al prodotto.

3.3 Gestione della qualità

3.3.1 Scopo

Lo scopo del processo di gestione della qualità è garantire che il prodotto finale rispetti i requisiti di qualità individuati dagli stakeholder$_G$, che le esigenze espresse dal proponente siano soddisfatte e che la qualità sia nel complesso soddisfacente. Si mira inoltre ad un maggior controllo di prodotti e processi.

3.3.2 Descrizione

Alla Gestione della qualità è dedicato il *Piano di Qualifica*, documento che descrive le metriche e le modalità usate per valutare il livello di qualità di prodotti e processi.
Tale documento si articola in:

- esposizione degli obiettivi di qualità dei processi;
- esposizione delle metriche di qualità dei processi con relativi valori soglia;
- esposizione degli obiettivi di qualità dei prodotti;
- esposizione delle metriche di qualità dei prodotti con relativi valori soglia;
- tracciamento obiettivo-metrica e metrica-obiettivo;
- specifiche dei test per il software;
- resoconti delle attività di verifica sui prodotti;
- considerazioni sul miglioramento;

3.3.3 Aspettative

Le aspettative di questo processo sono:

- conseguimento della qualità nel prodotto, secondo le richieste del proponente;
- prova oggettiva della qualità del prodotto;
- conseguimento della qualità nell'organizzazione delle attività del gruppo e dei processi;
- raggiungimento della piena soddisfazione del proponente.

3.3.4 Controllo di qualità

Stabilire delle regole di comportamento e di azioni che permettano il raggiungimento di un sistema di qualità predisposto al miglioramento continuo è fondamentale.
Le attività che ciascun membro deve svolgere per assicurare l'ottenimento della qualità desiderata sono:

1. comprensione, per ogni task, degli obiettivi da raggiungere;
2. individuazione di eventuali errori e discrepanze rilevate;
3. produzione di una stima in valore per ogni task da svolgere;
4. produzione di una stima in termini di dimensione e complessità per ogni task da svolgere;
5. impiego delle competenze di ciascun membro del gruppo;
6. produzione di risultati concreti e quantificabili;
7. miglioramento costante della propria formazione e capacità attraverso una stabile comunicazione di gruppo;
8. utilizzo del sistema di tiketing offerto da *GitHub*, per tracciare ogni task;
9. svolgimento di *Code refactoring*$_G$;
10. svolgimento di *Continuous Integration*$_G$;
11. rispetto totale degli standard e di quanto normato all'interno del suddetto documento;

3.3.4.1 Code refactoring

Il *code refactoring* consiste nel migliorare la struttura del codice, senza alterarne il comportamento esterno. Viene applicato per migliorare le caratteristiche non funzionali del software, così da aumentarne la qualità e agevolarne la manutenzione.
La ristrutturazione eseguita permette di:

- migliorare il design del codice;
- aumentare il grado di comprensione del codice;
- aumentare l'individuazione di bug/errori esistenti;
- permettere di sviluppare codice più velocemente.

Il *code refactoring* viene svolto da ciascun *Programmatore* solo quando il software ha passato tutti i test definiti all'interno del *Piano di Qualifica*.

Col *code refactoring*, si parla di *code smell*$_{\mathbf{G}}$, che prevede:

- la scelta di indicatori e nomi espressivi;
- l'eliminazione di complessità del sorgente;
- la cancellazione di codice ridondante, duplicato o superfluo.

Il refactoring non è esclusivamente a carico dei *Programmatori*; ad esempio, per il codice scritto in *Node.js* esso viene supportato in modo automatizzato con l'IDE$_{\mathbf{G}}$ di *Visual Studio*.

Bad Smells Code

Per *Bad Smells Code* si intendono le caratteristiche del codice sorgente indicatrici di un difetto di programmazione. Per garantire la qualità del prodotto software è importante non solo che il codice esegua correttamente ma anche che non siano presenti delle debolezze di design del codice stesso.
Esistono elenchi di numerosi smells code, il gruppo *ProApes* ha deciso di considerare:

- ***Bloaters***: codici, metodi e classi che, attraverso una crescita graduale, diventando troppo estesi, rendendo il lavoro più difficile da svolgere; problemi simili emergono solo con l'evolversi del programma, accumulandosi nel tempo.
- ***Object-oriented abusers***: procedure che non rispettano i principi della programmazione ad oggetti;
- ***Dispensables***: sezioni di codice senza alcuna utilità funzionale o strutturale, la cui presenza anzi rende il codice meno leggibile e manutenibile;
- ***Couplers***: codice che causa eccessivo accoppiamento tra classi;
- ***Change preventers***: sezioni di codice la cui modifica, anche solo puntuale e circoscritta, provoca a cascata la necessità di modificare anche altre parti di codice.

La trattazione di questi aspetti verrà approfondita in sede di *Progettazione di dettaglio*.

3.3.4.2 Continuous Integration

Nata per combattere la divergenza del codice, la*Continuous Integration* va applicata quando lo sviluppo software è affiancato ad un sistema di versionamento. Essa permette di allineare frequentemente, più volte al giorno, gli ambienti di lavoro dei vari sviluppatori verso l'ambiente di lavoro condiviso. I concetti e le tecnologie che concorrono alla creazione di tale pratica permettono di far fronte a tutte le problematiche dovute all'aggiornamento delle dipendenze nel codice.
È quindi un'attività basilare, che deve essere applicata da ogni membro del gruppo, in quanto premette di prevenire l'insorgenza di problemi legati all'incompatibilità, in modo che questi abbiano conseguenze drammatiche sul lavoro svolto.

Il gruppo *ProApes* ha optato per l'impiego di *GitHub Actions*$_{\mathbf{G}}$ come strumento di *Continuous Integration*. L'utente ha la possibilità di creare un workflow che esegua il codice presente nella repository ed effettui i test al verificarsi di un determinato evento (come ad esempio un push in un determinato branch). Tutte le impostazioni riguardanti il workflow sono memorizzate in un file *.yml* all'interno della cartella *.github/workflows*, e sono strutturati come descritto:

```
name: nome-workflow
on:
    push:
        branches:
         - brach-da-monitorare
jobs:
    build:
        runs-on: sistema-operativo
steps:
 - with:
    node-version: *
 - run: istruzioni-di-installazione
 - run: istruzioni-di-build
 - run: istruzioni-di-test
```

3.3.5 Istanziazione di un processo

Il gruppo ha deciso che, al fine di perseguire la qualità, questa deve essere curata e monitorata sin dall'inizio delle attività di progetto, partendo dall'istanziazione dei processi. A tal proposito, i membri di *ProApes* hanno deciso adottare le seguenti regole al momento della definizione di ogni processo:

- un singolo processo deve puntare a perseguire un obiettivo unico, così da ridurne la complessità;
- l'obiettivo di un processo non deve sovrapporsi agli obiettivi di altri processi, già istanziati o la cui istanziazione è già stata pianificata; in questo secondo caso, il gruppo si riserva di valutare, caso per caso, se rivedere la pianificazione o se modificare il processo in questione (causa dell'eventuale conflitto);
- quando si iniziano le attività di un nuovo processo, l'organizzazione delle risorse affidategli deve tenere conto degli altri processi in corso in quel momento, in modo da garantire uno sfruttamento ragionevole ed efficiente delle risorse umane, temporali e materiali;
- ad ogni processo deve corrispondere un'analisi preventiva dei rischi che esso potrebbe comportare e la pianificazione di strategie per contrastarli;
- al momento dell'istanziazione di un processo devono essere note la sua durata e di conseguenza la sua data di conclusione.

3.3.6 Denominazione degli obiettivi

Per la denominazione degli obiettivi di qualità, il gruppo ha deciso di adottare il formato

O[Categoria][TipoProdotto][X]

dove

- **O** indica che si tratta di un obiettivo di qualità;
- **[Categoria]** specifica a quale categoria appartiene l'obiettivo, e assume i valori:
  - **PR** per i processi;

  - **PD** per i prodotti.

- **[TipoProdotto]**, presente in caso di obiettivi riguardanti i prodotti, indica se si riferisce a documenti o software, e assume i valori:
  - **D** per i documenti;
  - **S** per i prodotti software.
- **[X]** indica l'identificativo numerico a due cifre per la metrica; la numerazione inizia da 1.

Tutti gli obiettivi presentati nella documentazione del progetto fanno fede a queste regole di denominazione.
Il gruppo *ProApes* si riserva di integrare l'elenco degli obiettivi esposto nel presente documento e nel *Piano di Qualifica 4.0.0-1.10* qualora, durante le attività successive del progetto, emergesse la necessità di stabilirne di nuovi.

3.3.7 Denominazione delle metriche

Per la denominazione delle metriche$_{\mathbf{G}}$ il gruppo ha deciso di adottare il formato

M[Categoria][TipoProdotto][X]

dove

- **M** indica che si tratta di una metrica di qualità;
- **[Categoria]** specifica a quale categoria appartiene la metrica, e assume i valori:
  - **PD** per i prodotti;
  - **PR** per i processi;
  - **TS** per i test.
- **[TipoProdotto]**, presente solo in caso di metriche di prodotto, indica se si riferisce a documenti o software, e assume i valori:
  - **D** per i documenti;
  - **S** per i prodotti software.
- **[X]** indica l'identificativo numerico a due cifre per la metrica; la numerazione inizia da 1.

Tutte le metriche esposte nella documentazione del progetto faranno fede a queste regole di denominazione.
Il gruppo *ProApes* si riserva di integrare l'elenco delle metriche esposto nel presente documento e nel *Piano di Qualifica 4.0.0-1.10* qualora, durante le attività successive del progetto, emergesse la necessità di adottarne di nuove.

3.3.8 Metriche adottate

La descrizione delle metriche per la qualità di processo e di prodotto selezionate dal team è riportata in §D.

3.4 Verifica

3.4.1 Scopo

Il processo di Verifica ha per obiettivo la realizzazione di prodotti corretti, coesi e completi. Sono soggetti a verifica sia il software che la documentazione.

3.4.2 Descrizione

Il processo di Verifica viene applicato su ogni processo in esecuzione:

- al raggiungimento di un livello di maturità sufficiente;
- successivamente a cambiamenti inerenti lo stato.

Per ciascun processo viene analizzata la qualità dei prodotti ottenuti e dei processi impiegati.

3.4.3 Aspettative

Il processo di Verifica prende in input ciò che è già stato prodotto e lo restituisce in uno stato conforme alle aspettative. Per ottenere tale risultato ci si affida a processi di analisi e test. Per assicurare il corretto svolgimento di tale processo si devono rispettare i punti seguenti:

- viene effettuato seguendo procedure definite;
- vi sono criteri chiari e affidabili da seguire per verificare;
- i prodotti passano attraverso fasi successive, ognuna delle quali è verificata;
- dopo la verifica il prodotto si trova in uno stato stabile;
- permette di procedere alla fase di validazione.

3.4.4 Verifica della documentazione

È compito del *Responsabile di Progetto* stabilire l'inizio delle attività di Verifica, pianificandone data d'inizio e durata e assegnando i ruoli di *Verificatore* ai membri del gruppo. Questi ultimi dovranno svolgere un'analisi accurata dei documenti assegnateli mirata a:

- controllo della sintassi;
- rispetto delle norme tipografiche stabilite all'interno del presente documento;
- utilizzo di frasi con periodi brevi;
- controllo della struttura del documento;
- controllo della pertinenza dei contenuti del documento.

Per adempiere in modo più agevole alle loro mansioni i *Verificatori* fanno uso di $Aspell_{\boldsymbol{G}}$.

3.4.4.1 Analisi statica

Questo tipo di analisi svolge un controllo sulla documentazione. Il *Verificatore* incaricato adotterà uno dei due metodi sotto presentati:

- ***Walkthrough***: tecnica dove il *Verificatore* eseguirà una lettura e un controllo completo dell'intero documento alla ricerca di eventuali errori. *Verrà utilizzato questo metodo fino a quando non si conosceranno gli errori potenzialmente riscontrabili;*
- ***Inspection***: tecnica in cui il *Verificatore* eseguirà una lettura e un controllo del documento mirato nei punti dove si sa già che possano essere presenti errori.

La tecnica *Walkthrough* risulta molto onerosa a causa dell'adozione di una ricerca a pettine. Per questo deve essere impiegata solo nelle prime fasi del progetto o quando non è possibile avere la lista degli errori comuni. Con l'incedere dell'attività di progetto, e quindi con il ripetersi del processo di Verifica sulla documentazione, si otterrà una lista degli errori comuni, denominata Lista di Controllo$_{\mathbf{G}}$, consentendo l'utilizzo del *Inspection*.

3.4.5 Procedimento di verifica della documentazione

Si riporta di seguito sotto forma di diagramma delle attività la procedura impiegata dal gruppo *ProApes* per la verifica dei documenti.

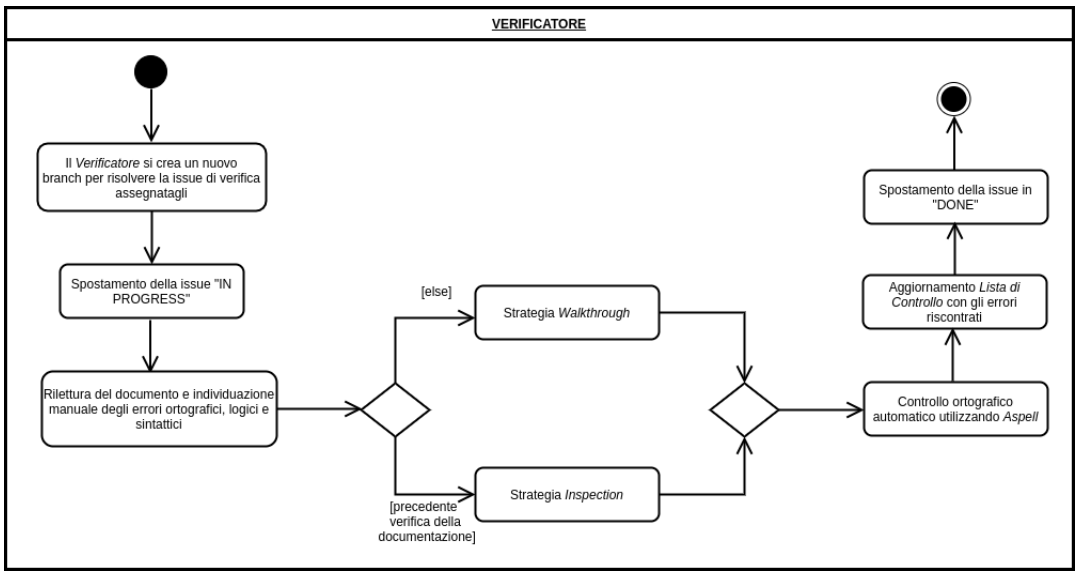

Figura 28: Diagramma delle attività di verifica della documentazione

3.4.6 Verifica del codice

Il codice verrà verificato con test automatici, appositamente creati, e tramite misurazioni quantificabili definite all'interno del *Piano di Qualifica 4.0.0-1.10*. Il codice dovrà quindi risultare conforme ai seguenti criteri:

- codice testabile, corretto e rispettoso di quando dichiarato dai requisiti e dallo standard di codifica adottato;
- interfacce e dati consistenti;
- codice derivato dalla progettazione o dai requisiti;
- corretta sequenza di operazioni implementate conformi agli eventi.

3.4.6.1 Analisi statica

L'analisi statica non è prerogativa unica della verifica dei documenti: coinvolge anche il codice prodotto.
Come strumento di automazione di tali attività, il gruppo ha scelto $ESLint_{\mathbf{G}}$. Tale software permette l'analisi statica del codice, è integrabile con moltissimi IDE di codifica, può effettuare in automatico alcune correzioni di errori rilevati e permette di personalizzare le regole secondo cui valuta il codice.

3.4.6.2 Analisi dinamica

L'analisi dinamica è una tecnica di verifica del codice generato, che viene testato durante la sua esecuzione. Su di esso vengono generati ed eseguiti una serie di test case, per verificarne il corretto funzionamento e per rilevare eventuali scostamenti tra i risultati ottenuti e quanto atteso.
Ogni test è decidibile, inoltre è fondamentale sia ripetibile: dato un certo input si deve ottenere sempre il medesimo output per ogni prova effettuata.
Ogni test deve presentare i seguenti parametri:

- **ambiente:** sistema hardware e software sul quale verrà eseguito il test;
- **stato iniziale:** stato iniziale dal quale il test viene eseguito;
- **input:** dati in ingresso immessi;
- **output:** dati in uscita attesi;
- **istruzioni aggiuntive:** ulteriori istruzioni su come deve essere eseguito il test e su come interpretare i risultati ottenuti.

Dei test ben scritti, per definirsi tali, devono:

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

3.4.7 Verifica dei requisiti

Anche i requisiti vengono sottoposti a Verifica; vengono applicate le strategie di *Walkthrough* ed *Inspection* per controllarne la validità.
Per soddisfare positivamente la verifica essi devono:

- presentare coerenza con quanto dichiarato al proponente e descritto all'interno dell'*Analisi dei Requisiti 4.0.0-1.10*;
- dimostrarsi sempre verificabili e quantificabili;
- essere lo specchio di quanto implementato, sotto forma di codifica, nel sistema;
- avere con un livello di fattibilità adeguato rispetto al tempo e alle risorse pianificate.

3.4.7.1 Analisi statica

I requisiti devono presentare le caratteristiche di:

- **verificabilità**: un requisito si deve presentare quantificabile e misurabile oggettivamente;
- **codice univoco**: un requisito deve essere identificato da un codice, differente da requisito a requisito e rispettare quanto dichiarato in §2.2.4.5;
- **atomicità**: un requisito deve essere indivisibile.

3.4.8 Test

I test rappresentano l'attività fondamentale dell'analisi dinamica: dimostrano che effettivamente il programma svolge i tasks per cui è stato sviluppato. Permettono di individuare tutti i possibili errori che possono essere stati commessi prima della messa in uso sul mercato del prodotto software e di valutarne la conformità rispetto all'*Analisi dei Requisiti*.
Saranno sviluppati i seguenti test, ognuno dei quali ha un diverso oggetto di verifica e scopo.

3.4.8.1 Test d'unità

I test d'unità verificano la correttezza degli algoritmi. L'obiettivo principale di questa metodologia di test consiste nell'isolare la parte più piccola di software testabile, chiamata unità, in modo da poterne stabilire il corretto funzionamento rispetto alle attese. Ogni singola unità deve essere sottoposta a test prima di procedere alla sua integrazione. Le singole unità possono essere testate con l'ausilio di driver$_{\mathbf{G}}$, simulatori di chiamate da parte dell'utente o da parte del sistema, e stub$_{\mathbf{G}}$, simulatori di un'altra unità.

3.4.8.2 Test d'integrazione

I test d'integrazione verificano la correttezza delle interfacce. Rappresentano l'estensione logica dei test d'unità e sono i predecessori dei test di sistema. Il principio di questa categoria di test è aggregare mano a mano tutte le varie componenti del software prodotto, verificandone la sinergia e valutando il funzionamento del sistema nel suo complesso. Ogni test prevede vi siano almeno un paio di componenti correlate per sottoporre a verifica tutte le unità che compongono un processo.

3.4.8.3 Test di sistema

I test di sistema testano l'applicazione nella sua interezza. Viene effettuato un controllo sulle componenti del sistema che devono risultare:

- compatibili fra loro;
- con proprie interazioni e scambio di dati tra le interfacce, conformi ed al momento appropriato.

Si portano così alla luce le ipotesi errate che i diversi sviluppatori hanno fatto sulle parti di software sviluppate da altri.
I test di sistema caratterizzano la validazione del prodotto software finale, mezzo con il quale si verifica che tutti i requisiti siano soddisfatti in modo completo.

3.4.8.4 Test di regressione

I test di regressione si effettuano a seguito di modifiche al sistema, per verificare la qualità di nuove versioni del prodotto software. Lo scopo principale è valutare le nuove modifiche/funzionalità non ancora testate, introdotte nel codice, garantendo che quelle preesistenti abbiano mantenuto le proprie caratteristiche qualitative. Risulta quindi necessaria la riesecuzione dei test sul codice modificato, per poter stabilire in modo esatto se vi è stata un'alterazione del comportamento degli elementi precedentemente funzionanti a seguito delle modifiche introdotte.

3.4.9 Procedimento di verifica del codice

Si riporta di seguito sotto forma di diagramma delle attività la procedura impiegata dal gruppo *ProApes* per la verifica del codice.

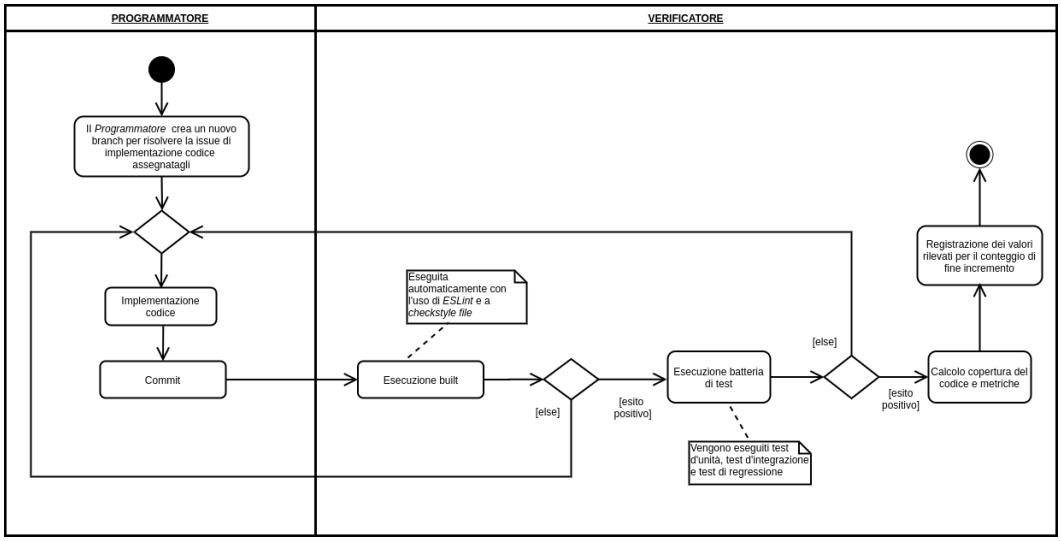

Figura 29: Diagramma delle attività di verifica del codice

3.4.10 Strumenti

Gli strumenti che il gruppo ha scelto di utilizzare per la verifica automatica del codice sono:

ESLint

Strumento di analisi statica del codice pensato per identificare pattern problematici ricorrenti all'interno di un progetto *JavaScript* o *TypeScript*. Può essere configurato e personalizzato, ed è in grado di identificare sia problematiche di qualità del codice, che di stile.

Jest

Framework di test *JavaScript* e *TypeScript* gestito da *Facebook*, utilizzato per effettuare test dinamici sul codice, in particolare test di unità e integrazione. Ha un particolare focus sulla semplicità di utilizzo, infatti è completamente personalizzabile e configurabile tramite file di configurazione direttamente nella cartella di lavoro.

SonarCloud

Servizio online *open-source* per la qualità e sicurezza del codice. Supporta il linguaggio *TypeScript*, si integra con la piattaforma *GitHub Actions* ed effettua analisi dinamica del codice ad ogni push. Restituisce in forma di dashboard metriche e informazioni inerenti la qualità del progetto testato.

3.5 Validazione

3.5.1 Scopo

Il processo di Validazione stabilisce se il prodotto è in grado di soddisfare il compito per il quale è stato creato. Successivamente alla Validazione, è garantito che il software rispetti i requisiti e che soddisfi i bisogni del committente.

3.5.2 Descrizione

Tale processo prende in esame il prodotto della fase di Verifica e lo restituisce con la garanzia che esso sia conforme ai requisiti e bisogni del committente.
Sarà il *Responsabile di Progetto* che avrà la responsabilità di controllare i risultati decidendo se:

- accettare ed approvare il prodotto;
- rigettare il documento, chiedendo un'ulteriore verifica del prodotto con nuove indicazioni.

3.5.3 Aspettative

Dovono essere rispettati i seguenti punti:

- identificazione degli oggetti da validare;
- identificazione di una strategia con procedure di validazione tali per cui le procedure di verifica siano riutilizzabili;
- applicazione della strategia;
- valutazione dei risultati rispetto alle attese.

3.5.4 Procedimento di validazione della documentazione

Di seguito viene mostrato il diagramma delle attività esplicativo del processo di Validazione da applicare a tutti i documenti prodotti dal gruppo *ProApes* durante il progetto.

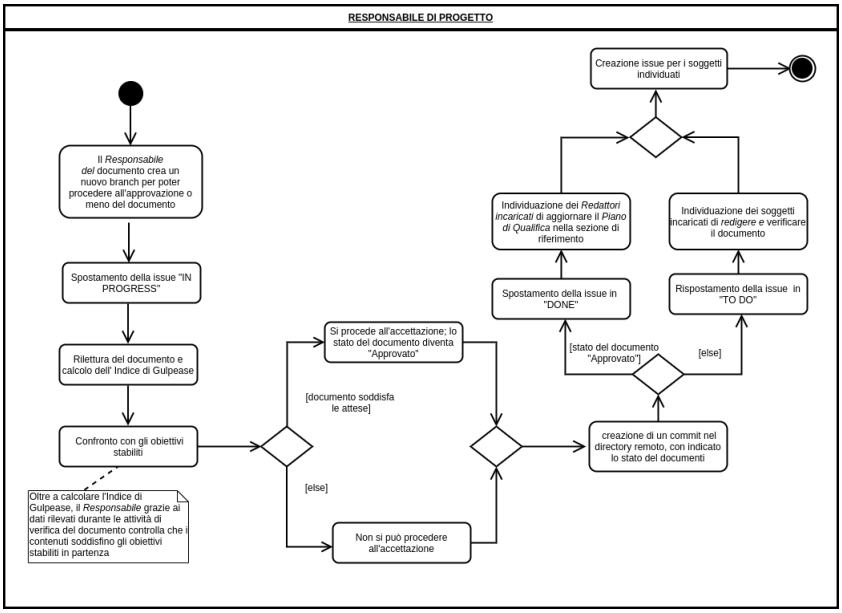

Figura 30: Diagramma delle attività di validazione della documentazione

3.5.5 Attività per la Validazione

La validazione del codice si compone di due attività fondamentali:

- **test di validazione e di sistema:** l'insieme delle attività eseguite internamente al gruppo;
- **collaudo:** attività di controllo che si svolge con la presenza di un committente, durante la *Revisione di Accettazione*. Dimostra la conformità del prodotto software secondo quanto promesso da contratto nei casi d'uso e nei requisiti espressi dall'*Analisi dei Requisiti 4.0.0-1.10*;

I test d'integrazione permettono l'esecuzione dei *test di sistema*, svolgendo un controllo sulle funzionalità implementate. Questo permette di stabilire con certezza se si sta rispettando quanto dichiarato nel contratto con il proponente.

Il tracciamento$_{\mathbf{G}}$ risulta predominante per poter confermare il pieno rispetto e soddisfacimento di tutti i requisiti. Il processo di Validazione prevede le seguenti attività:

- **pianificazione e tracciamento** dei test da eseguire sul codice prodotto;
- **esecuzione dei test**, dove vengono create situazioni di forzatura del sistema nei suoi punti critici per evidenziarne gli errori;
- **aggiornamento** del *Piano dei Test*;
- **controllo dei risultati**, dove viene verificato se il sistema soddisfi i requisiti.

All'interno del *Piano di Qualifica* si include il *Piano dei Test*; tale documento definisce l'esecuzione dei test di sistema e di validazione.
Per ogni test eseguito devono venire specificati:

- le entità sottoposte a validazione;
- i task e gli obiettivi della validazione;

- i *Verificatori* che hanno eseguito il test e le risorse impiegate;
- resoconto con identificazione dei risultati ottenuti, attesi e problematiche insorte durante l'esecuzione del test.

3.5.5.1 Test di validazione

Anche detto "test di collaudo", è simile al test di sistema per l'oggetto testato, ma viene eseguito con la partecipazione dei committenti. Verifica il prodotto e, in particolare, il soddisfacimento del cliente.
Il superamento del test di collaudo garantisce che il software è pronto per essere rilasciato.

3.5.5.2 Responsabilità dei test

Tre sono i soggetti coinvolti nell'ambito dei test di validazione: *Responsabile di Progetto*, *Verificatori* e *Progettisti*. Nello specifico:

- ***Progettisti***: pianificano e implementano i test; per assicurare una natura dei test non condizionata ed imparziale rispetto al codice scritto ed una certa obiettività nell'esecuzione degli stessi, tali figure non devono coincidere con coloro che hanno progettato o implementato la porzione di codice da testare;
- ***Verificatori***: eseguono il test sul prodotto software, ne tracciano i risultati e aggiornano il *Piano dei Test*.
- ***Responsabile di Progetto***: controlla quanto descritto nel *Piano dei Test* e valuta i risultati ottenuti.

  Sarà il *Responsabile di Progetto* a decidere se procedere con:

  - **accettazione**, quando il risultato ottenuto dal test risulta sufficientemente buono;
  - **ripetizione**, ad opera dei *Verificatori* con eventualmente la sua presenza; quando il risultato dei test presenta un margine tra i risultati ottenuti e quelli attesi troppo elevato.

3.5.5.3 Codice identificativo dei test

Ogni test viene descritto con:

- codice identificativo;
- descrizione;
- stato, che può assumere uno dei seguenti valori:
  - implementato;
  - non implementato;
  - non eseguito;
  - superato;
  - non superato.

Ogni test è definito con l'ausilio del codice sottostante:

T[Tipo][ID]

dove

- **T:** valore statico identificativo di "Test";
- **Tipo:** rappresenta un valore statico. Assume uno fra i valori letterali:
  - **U:** test d'unità;

  - **I:** test d'integrazione;
  - **S:** test di sistema;
  - **R:** test di regressione;
  - **V:** test di validazione.

- **ID:** rappresenta un codice identificativo. I valori che può assumere sono:
  - **IDNumerico:** la cui applicazione avviene nel caso di test d'unità, d'integrazione e di regressione. Viene espresso con un codice numerico progressivo che inizia da 1;
  - **IDRequisito:** da usarsi per i test di accettazione e di sistema. Composto da:

    [Tipologia][Importanza][Codice]

    (come stabilito in §2.2.4.5), identifica il codice del requisito presente all'interno del documento *Analisi dei Requisiti*.

3.5.6 Procedimento di validazione del codice

Di seguito viene mostrato il diagramma delle attività esplicativo del processo di Validazione da applicare al prodotto software ralizzato dal gruppo *ProApes* durante il progetto.

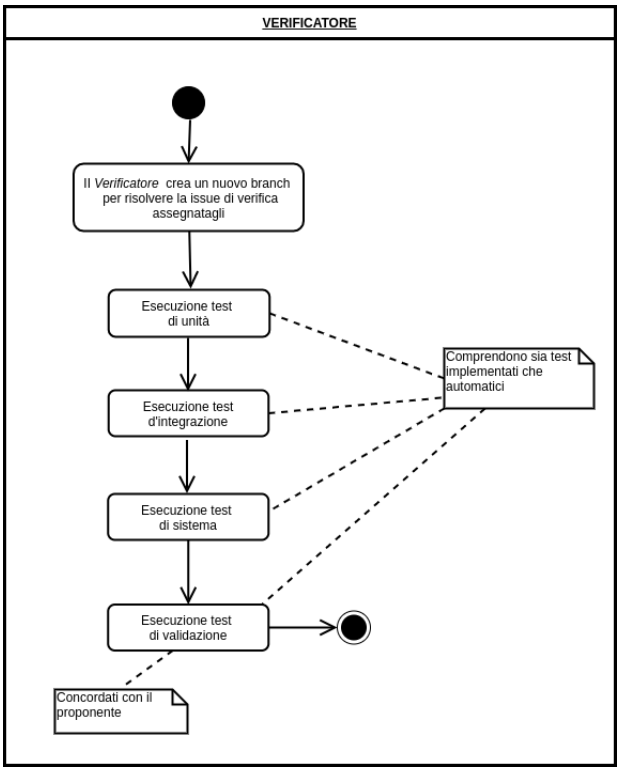

Figura 31: Diagramma delle attività di validazione del codice

3.6 Gestione dei cambiamenti

3.6.1 Scopo

Lo scopo del processo è garantire un mezzo tempestivo, affidabile e documentato per la gestione appropriata di tutti i problemi rilevati e il rilevamento di tendenze nello svolgimento delle varie attività.

3.6.2 Descrizione

Il presente processo consiste nell'analisi e soluzione di problemi riscontrati durante l'implementazione delle attività di progetto, di qualunque natura essi siano; le attività previste includono l'individuazione e gestione di non conformità del prodotto alle attese.

3.6.3 Aspettative

Grazie all'implementazione del processo di *Gestione dei cambiamenti*, il team si aspetta di essere sempre preparato di fronte a complicazioni o malfunzionamenti imprevisti, e di saper elaborare in tempi brevi una risposta adeguata ai problemi rilevati.

3.6.4 Prassi generale

Assegnazione, svolgimento e verifica di ogni attività o compito vengono normati dal ciclo di vita del ticket$_{\mathbf{G}}$ descritto in §4.1.5.6.
Al termine dello svolgimento di un'attività, essa viene sempre revisionata. Se durante l'attività di verifica vengono individuati degli errori, il *Verificatore* apre una issue$_{\mathbf{G}}$ di correzione, inserendovi un commento con le osservazioni sul lavoro revisionato e le correzioni da applicarvi; questa issue verrà poi assegnata al membro del gruppo che aveva svolto il compito in prima istanza: egli dovrà applicare quanto segnalato nei tempi più brevi possibili.
Se le correzioni segnalate dal *Verificatore* non risultano chiare, o l'esecutore del compito non le condivide, è previsto un confronto tra i due, che può essere indifferentemente svolto in presenza, per iscritto (tramite gli appositi canali) o in chiamata.
Qualora non si raggiungesse un accordo nemmeno con questi mezzi, viene coinvolto il *Responsabile di Progetto*, che, dopo essersi consultato con le due parti, deciderà definitivamente la soluzione da adottare.

3.6.5 Denominazione dei cambiamenti

Ai fini della tracciabilità, ad ogni cambiamento viene assegnato un codice secondo lo schema

C[P/A][Attività/TipoProdotto][X]

dove:

- **C** indica che si tratta di un cambiamento;
- **P/A** assume il valore:
  - **P** se riguarda un prodotto;
  - **A** se riguarda un'attività di processo.
- **Attività/TipoProdotto**
  - se si tratta di un'attività di processo assume i valori:
    - **AN** se riguarda attività di Analisi;
    - **PG** se riguarda attività di Progettazione;
    - **CD** se riguarda attività di Codifica;
    - **ST** se riguarda attività di Gestione degli strumenti;
    - **DO** se riguarda attività di Documentazione;
    - **CG** se riguarda attività di Configurazione;
    - **QU** se riguarda attività di Gestione della qualità;
    - **VE** se riguarda attività di Verifica;
    - **CA** se riguarda attività di Gestione dei cambiamenti;
    - **CO** se riguarda attività di Coordinamento;
    - **PN** se riguarda attività di Pianificazione.

  - se si tratta di un prodotto assume i valori:
    * **DOC** se riguarda un prodotto documentale;
    * **COD** se riguarda un prodotto di codifica;

- **X** indica il numero del cambiamento all'interno della sua categoria; parte da 1.

3.6.6 Grado di priorità

Ad ogni cambiamento registrato viene assegnata una priorità, al fine di ordinare il processo di gestione in caso si verifichino più cambiamenti in breve tempo.
I gradi di priorità previsti sono:

- **A**: alto, corrisponde alla massima priorità, da gestire il prima possibile;
- **M**: medio, corrisponde alla priorità media, da gestire in tempi brevi ma senza urgenza;
- **B**: basso, corrisponde alla priorità più bassa, gestibile in tempi più lunghi;

4 Processi Organizzativi

4.1 Gestione di processo

4.1.1 Scopo

Il processo$_{\mathbf{G}}$ di Gestione, come stabilito dallo standard$_{\mathbf{G}}$ ISO/IEC 12207:1995$_{\mathbf{G}}$, contiene le attività$_{\mathbf{G}}$ e i compiti$_{\mathbf{G}}$ generici, da utilizzare per la gestione dei rispettivi processi.
Il *Responsabile di Progetto* è responsabile per:

- il risultato della gestione;
- la gestione del progetto stesso;
- la gestione dei compiti del processo o dei processi ad esso applicabili (e.g: Fornitura, Sviluppo, ecc).

4.1.2 Descrizione

Le attività di Gestione di processo sono:

- inizio e definizione dello scopo;
- istanziazione dei processi;
- pianificazione e stima di tempi, risorse e costi;
- assegnazione dei ruoli e dei compiti;
- esecuzione e controllo;
- revisione e valutazione periodica delle attività.

4.1.3 Aspettative

Le aspettative per questo processo sono:

- ottenere una pianificazione ragionevole delle attività da seguire;
- coordinare i membri del team assegnando loro ruoli, compiti e facilitando la comunicazione tra loro;
- adoperare processi per regolare le attività e renderle economiche;
- mantenere il controllo sul progetto in modo efficace ma non invasivo, monitorando il team, i processi e i prodotti.

4.1.4 Coordinamento

4.1.4.1 Scopo

L'attività di Coordinamento permette di gestire la comunicazione sia interna che esterna e gli incontri. Questi ultimi possono essere svolti esclusivamente tra i membri del gruppo *ProApes* oppure tra il team e soggetti esterni (proponente$_{\mathbf{G}}$, committenti$_{\mathbf{G}}$, competitors$_{\mathbf{G}}$ ed esperti esterni).

4.1.4.2 Aspettative

Tale attività permette di definire dei metodi di comportamento precisi ai quali ogni membro del gruppo si deve attenere durante il periodo di svolgimento del progetto.

4.1.4.3 Descrizione

Nella presente sezione il Coordinamento viene strutturato nel modo che segue:

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

4.1.4.4 Comunicazioni

Durante lo svolgimento del progetto il gruppo *ProApes* comunicherà su due livelli distinti: interno ed esterno. Per quanto riguarda le comunicazioni esterne, esse avverranno con i seguenti soggetti:

- **Proponente**: l'azienda *Zucchetti S.P.A*, rappresentata dal *Dott. Gregorio Piccoli*;
- **Committenti**: nella persona del *Prof. Tullio Vardanega* e del *Prof. Riccardo Cardin*;
- **Competitors**: i gruppi *Carbon12* e *VRAM Software*. Tali soggetti analogamente a *ProApes* appartengono ai gruppi del primo lotto che lavorano al progetto *Predire in Grafana*;
- **Esperti interni**: da consultare eventualmente previo accordo con il proponente ed i committenti.

Il gruppo si rivolgerà al/ai competitors informalmente di persona, mentre intratterrà comunicazioni scritte e/o riunioni con il proponente, i committenti e gli esperti.

Comunicazione interna

Il mezzo di comunicazione standard per interazioni scritte tra membri del gruppo *ProApes* sarà *Slack*$_{\mathbf{G}}$, servizio di messaggistica istantanea rivolto alla collaborazione aziendale. La politica per la gestione delle discussioni sarà inoltre la seguente: per ogni compito di importanza non trascurabile che il gruppo dovrà svolgere verrà creato un canale specifico (ad esempio tutte le discussioni riguardanti un documento avverranno all'interno del canale apposito). Oltre a questo tipo di canali, destinati a mutare nel corso del tempo, le discussioni verranno raggruppate nel seguente modo:

- **git-github**: per qualsiasi discussione e/o problemi riguardanti le repository$_{\mathbf{G}}$;
- **predire-in-grafana**: per qualsiasi discussione e/o problemi riguardanti il capitolato$_{\mathbf{G}}$ C4;
- **general**: per qualsiasi discussione generica o riguardante il progetto;
- **resoconto-giornaliero**: da utilizzare al termine di ogni giornata, notifica all'intero team quanto fatto da ciascuno a partire dall'ultimo aggiornamento.

Includendo nel corpo le seguenti parole chiave, *Slack* permette di notificare un particolare messaggio a una o più persone:

- **@everyone**: il messaggio verrà notificato a tutti i componenti;
- **@channel**: il messaggio verrà notificato a tutti gli utenti iscritti a quel particolare canale;
- **@username**: inserendo l'username specifico, il messaggio verrà notificato all'utente desiderato.

In situazioni eccezionali, quali l'impossibilità di usare *Slack*, i membri del gruppo possono fare ricorso a servizi di comunicazione alternativi (ad esempio *Telegram*$_{\boldsymbol{G}}$ o SMS) per non dover sospendere il proseguimento del lavoro.

Esistono infine altri due tipi di comunicazioni interne:

- **comunicazioni orali**: i membri del gruppo potranno discutere informalmente di argomenti riguardanti il progetto, per chiarire dubbi o proporre idee;
- **comunicazioni tramite videochiamata**: per le videochiamate il gruppo utilizzerà *Discord*$_{\boldsymbol{G}}$, preferito ad altri servizi per la sua versatilità, per il fatto di essere open-source$_{\mathbf{G}}$ e perché multi-piattaforma.

Comunicazione esterna

Sarà il *Responsabile di Progetto*, a nome dell'intero gruppo, a mantenere i contatti verso l'esterno attraverso un indirizzo di posta elettronica creato appositamente. L'e-mail utilizzata sarà:

proapes11@gmail.com

Lo stesso *Responsabile di Progetto* dovrà notificare a ogni membro del gruppo di eventuali corrispondenze pervenute dai committenti e proponenti applicando le norme relative alle comunicazioni interne definite nella sezione precedente.
È stato deciso che ogni e-mail inviata presenterà la seguente struttura:

- **Oggetto**: l'oggetto della mail sarà preceduto dalla sigla "[SWE][UNIPD]", per rendere subito chiaro l'ambito di riferimento della mail stessa; esso dovrà essere espresso nella maniera più chiara e concisa possibile, per eliminare eventuali ambiguità e favorire la comprensione dell'argomento;
- **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 *Zucchetti S.p.A.*, nella persona di", "Gentile" o "Spettabile", dipendentemente dal destinatario. Il contenuto dovrà inoltre essere sintetico ed esaustivo, per esporre al meglio problemi e/o le eventuali richieste.

Quando necessario sarà anche possibile effettuare delle videochiamate attraverso l'impiego di *Hangouts*$_{\boldsymbol{G}}$.

4.1.4.5 Riunioni

Vengono definite qui le modalità di svolgimento delle riunioni, interne o esterne. Per ogni riunione sarà nominato un segretario, che terrà traccia delle discussioni affrontate, farà rispettare l'ordine del giorno ed infine redigerà un **verbale di riunione** (vedi §3.1.7.3) usufruendo delle informazioni raccolte.

Riunioni interne

La partecipazione alle riunioni interne sarà permessa ai soli membri del gruppo. Alla riunione dovranno partecipare almeno quattro (4) persone perché possa esserne riconosciuta la validità; le decisioni verranno infatti prese per maggioranza semplice. Inoltre, perché una riunione sia ritenuta valida, il *Responsabile di Progetto* dovrà portare a termine i seguenti compiti:

- fissare preventivamente la data della riunione, previa verifica della disponibilità dei membri del gruppo;
- stabilire un ordine del giorno;
- comunicare data e ordine del giorno, oppure avvertire con un anticipo adeguato se vi sono ritardi, modifiche o cancellazioni;

- nominare un segretario per la riunione;
- approvare il verbale della riunione.

I partecipanti alla riunione, invece, devono:

- avvertire preventivamente in caso di assenze o ritardi;
- presentarsi puntuali all'ora e luogo stabiliti;
- partecipare attivamente alla discussione.

Ogni membro del gruppo avrà la possibilità di richiedere un incontro interno: tale domanda dovrà essere indirizzata solo al *Responsabile di Progetto*, il quale pianificherà un eventuale incontro se necessario.

Riunioni esterne

Le riunioni esterne prevedono la partecipazione di soggetti esterni, quali il proponente e i committenti, oltre ai componenti del gruppo *ProApes*.
Così come le riunioni interne, anche le riunioni esterne prevedono la nomina di un segretario che dovrà poi redigere un verbale di riunione. Le riunioni esterne con il proponente avverranno nell'ufficio dell'azienda *Zucchetti S.p.A.* (Via Cittadella 7, 35121 Padova (PD)). Se invece si presenterà la possibilità, verranno utilizzate le aule di Torre Archimede - Dipartimento di Matematica "Tullio Levi Civita" (Via Trieste 63, 35121 Padova (PD)), previa disponibilità.

4.1.4.6 Lavorare a distanza

Vista la situazione in cui versa il nostro Paese e le regole di isolamento sociale che tutti siamo tenuti a rispettare, imposte a partire da inizio marzo 2020, la presente sezione espone il comportamento e le regole a cui ogni membro del gruppo si deve attenere fino alla conclusione delle limitazioni vigenti. Si è provveduto sia a disciplinare le situazioni nuove sorte a seguito di tali circostanze, sia a rendere più stringenti regole già esistenti in quanto relative a circostanze che possono risultare maggiormente delicate senza l'incontro di persona.
Tale normazione si auspica di permettere al team di proseguire nella realizzazione del progetto *Predire in Grafana* garantendo anche se non in presenza, una comunicazione e un organizzazione costante, chiara ed omogenea del lavoro da svolgere.

Incontri di gruppo

Gli incontri interni fra i membri del gruppo verranno svolti a cadenza settimanale, o più frequentemente se ce ne fosse necessità; il giorno prescelto verrà stabilito di comune accordo fra tutti i membri. Le modalità rimangono le stesse stabilite per le riunioni interne in presenza (§4.1.4.5 nella sotto-sezione relativa), con la variante di essere svolte esclusivamente attraverso videochiamate. Il giorno prima sul canale *Slack* dedicato agli incontri, il *Responsabile di Progetto* pubblicherà un documento riassuntivo dell'ordine del giorno per l'indomani; ogni membro del gruppo è tenuto a visionarlo, e nel caso se ne evidenziasse l'esigenza, dopo averne discusso con tutti, ad integrarlo. La piattaforma preferenziale da impiegare per tali riunioni "telematiche" resta *Discord*, seguita a ruota nel caso di malfunzionamenti da *Hangouts* e *Skype*.

Incontri con proponente e committenti

Gli incontri con proponente e committenti verranno svolti analogamente al punto precedente attraverso strumenti di videochiamata. Il gruppo lascerà ai soggetti esterni la scelta della piattaforma sulla quale svolgere l'incontro, in modo da permettere a questi ultimi di interfacciarsi con ciò che ritengono migliore alle proprie esigenze. Solo nel caso venga espressamente richiesto sarà il gruppo a scegliere lo strumento da impiegare. Se l'incontro non

fosse realizzabile (e.g: problemi di connessione, impossibilità dei proponenti/committenti/-del gruppo) si opterà per comunicazioni via e-mail come trattato nelle comunicazioni interne in §4.1.4.4, così da arginare il più possibile eventuali ritardi nello svolgimento del lavoro. Il gruppo cercherà comunque di mettersi al più presto in contatto con proponenti e committenti (stabilendo nuove date per gli incontri, trovando nuove modalità di comunicazione ecc.).

Reperibilità

Ciascun membro del gruppo si impegna ad essere il più reperibile possibile sia nei confronti del proprio team che dei soggetti esterni coinvolti nel progetto. Ogni membro deve fornire al *Responsabile di Progetto*, a intervalli regolari, prove di quanto da lui svolto. Questo già accade senza le limitazioni sociali del periodo, ma ora non avendo la possibilità di incontrarsi di persona queste dimostrazioni devono venire attuate in modo più puntuale: evidenziandone se ve ne sono, le difficoltà riscontrate nel coordinarsi/comunicare col team. Il *Responsabile di Progetto* in risposta provvederà a tenere sotto costante controllo l'andamento del lavoro garantendo una comunicazione e assegnazione dei compiti che sia il più efficiente possibile. Nel caso a uno dei membri del team venga meno uno degli strumenti di comunicazione stabiliti, il gruppo provvederà a ricercarne di nuovi il più velocemente possibile.

Problemi di salute e/o famigliari

Ogni membro del gruppo deve prontamente notificare al *Responsabile di Progetto* l'insorgere di eventuali problemi di salute e/o famigliari che vadano a intaccare il suo naturale proseguimento del lavoro all'interno del progetto. Il *Responsabile di Progetto* provvederà a redistribuire i compiti tra i membri restanti nel modo più omogeneo possibile. Sarà inoltre sempre compito del *Responsabile di Progetto* andare a monitorare la situazione sia del membro coinvolto che del resto del team; nel caso tali difficoltà persistano nel tempo si provvederà a modificare quanto pianificato dandone comunicazione ai committenti.

Difficoltà nello svolgimento dei compiti

Nel caso di difficoltà riguardanti sia l'organizzazione dei compiti (e.g: un compito si rivela eccessivamente impegnativo per una singola persona), sia l'attuazione pratica del compito stesso per carenza o mancanza di conoscenze da parte di un componente del team, esse devono essere prontamente comunicate al *Responsabile di Progetto*. Laddove il compito risulti troppo oneroso per una singola persona esso verrà frammentato in più sotto-compiti; nel caso invece di carenze conoscitive del soggetto interessato, ogni membro del gruppo cercherà di rendersi maggiormente disponibile a chiarimenti attraverso l'impiego di *Discord* e/o *Slack*. Risultando più aperti al dialogo e al confronto si spera così di evitare di incorrere in errori e ritardi nelle scadenze evitabili con una giusta comunicazione interna. Se necessario attraverso videochiamate si cercherà di aiutare il membro in difficoltà a svolgere i propri compiti; quest'ultimo provvederà non solo a sanare le proprie lacune attraverso uno studio individuale e un confronto continuo coi colleghi, ma si terrà in costante aggiornamento col *Responsabile di Progetto* in merito ai progressi fatti. Nel caso tale situazione non accenni a risolversi quest'ultimo provvederà a convocare una riunione fra i membri del gruppo per cercare le motivazioni del persistere di tale situazione. Nel caso peggiore si provvederà ad una redistribuzione dei compiti all'interno del team.

4.1.5 Pianificazione

4.1.5.1 Scopo

In accordo con lo standard ISO/IEC 12207$_{\mathbf{G}}$, è compito del *Responsabile di Progetto* predisporre l'attività di Pianificazione stabilendone la fattibilità e assicurandone l'esecuzione. Egli deve controllare che le risorse richieste per l'esecuzione e la gestione del processo

(in termini di personale, materiale, tecnologie e infrastruttura) siano disponibili, adeguate, appropriate e che il tempo a disposizione sia consono al raggiungimento degli obiettivi.

Il *Responsabile di Progetto* definirà nel *Piano di Progetto 4.0.0-1.10* i tempi e i modi per l'esecuzione del processo. Tale documento conterrà una descrizione delle attività e dei compiti correlati e consentirà l'identificazione degli obiettivi da raggiungere; strumenti e tecnologie da impiegare saranno invece elencati in *Norme di Progetto 4.0.0-1.10*.

Il *Piano di Progetto 4.0.0-1.10* comprende:

- una pianificazione in termini di calendario temporale sul completamento dei compiti;
- una stima dello sforzo richiesto dai membri del gruppo, espresso in termini di tempo;
- una presentazione delle risorse adeguate necessarie per eseguire i compiti;
- l'allocazione dei compiti;
- l'assegnazione delle responsabilità;
- una quantificazione dei rischi associati ai compiti, al processo e ai periodi di riferimento;
- una stima dei costi associati all'esecuzione del processo, correlati ai rispettivi periodi.

Misure a controllo della qualità da impiegare durante il processo saranno invece definite in *Norme di Progetto 4.0.0-1.10*. Per ogni periodo il *Responsabile di Progetto* dovrà fissare preventivamente delle milestone$_{\mathbf{G}}$ raggiungibili. Esse rappresenteranno il punto da cui partire per effettuare successivi incrementi, fornendo inoltre dei feedback di controllo sull'andamento del processo.

4.1.5.2 Descrizione

In questa sezione l'attività di Pianificazione, viene così articolata:

- ruoli di progetto;
- assegnazione dei compiti;
- ciclo di vita del ticket;
- gestione dei rischi.

4.1.5.3 Aspettative

La Pianificazione cerca di chiarire come il gruppo *ProApes* intende pianificare il lavoro da svolgere, a partire dalla scelta dei ruoli, sino alla concreta assegnazione dei compiti ai vari partecipanti. Verranno inoltre esposte le modalità di gestione dei rischi.

4.1.5.4 Ruoli di progetto

Nel corso del progetto, i componenti del gruppo ricopriranno i seguenti ruoli:

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

Essi corrispondono alle rispettive figure aziendali, e sarà stato stabilito un calendario che permetterà ad ogni membro di ricoprire almeno una volta ciascun ruolo per un periodo di tempo omogeneo, senza però gravare sullo svolgimento delle attività. Inoltre, tramite scelte oculate, si cercherà di eliminare eventuali conflitti di interesse: ad esempio, un componente non potrà redigere e poi verificare ciò che ha prodotto lui stesso.

Responsabile di Progetto

Il *Responsabile di Progetto* è un ruolo fondamentale, presente per tutta la durata del lavoro. Rappresenta il gruppo *ProApes* presso il proponente ed i committenti, ed il suo principale compito è coordinare l'intera struttura e organizzare il lavoro, in maniera tale che ogni sforzo fatto sia utile al fine comune.
I seguenti punti chiariscono nel dettaglio cosa questo ruolo comporta:

- deve occuparsi del coordinamento dei membri del gruppo e dei compiti che devono portare a termine (vedi §4.1.5.5);
- deve gestire la pianificazione, intesa come attività da svolgere e scadenze da rispettare;
- è responsabile della stima dei costi e dell'analisi dei rischi;
- deve curare le relazioni che il gruppo intratterrà con i soggetti esterni (vedi §4.1.4.4, §4.1.4.5 nelle specifiche sotto-sezioni);
- si occupa delle risorse umane e dell'assegnazione dei ruoli;
- deve approvare la documentazione.

Amministratore di Progetto

L'*Amministratore di Progetto* è incaricato di gestire, controllare e curare gli strumenti che il gruppo utilizza per svolgere il proprio lavoro. È la figura che garantisce l'affidabilità e l'efficacia$_{\mathbf{G}}$ dei mezzi scelti dal gruppo, perseguendo l'idea che la buona gestione dell'ambiente di lavoro (inteso come insieme di regole, strumenti e servizi), favorisca la produttività. Tale figura deve:

- amministrare infrastrutture e servizi necessari ai processi di supporto;
- gestire il versionamento e la configurazione dei prodotti;
- gestire la documentazione di progetto, controllando che venga corretta, verificata ed approvata, e facilitandone il reperimento;
- correggere eventuali problemi relativi alla gestione dei processi;
- redigere e manutenere norme e procedure che regolano il lavoro;
- individuare strumenti utili all'automazione dei processi.

Analista

L'*Analista* partecipa al progetto soprattutto nelle sue fasi iniziali, in particolare al momento della stesura dell'*Analisi dei Requisiti*; ha il compito di evidenziare i punti chiave del problema in questione, comprendendone appieno tutte le sue peculiarità. La sua figura è quindi fondamentale per la buona riuscita del lavoro, in quanto errori o mancanze nell'individuazione dei requisiti$_{\mathbf{G}}$ da soddisfare possono compromettere fortemente la successiva attività di Progettazione. Egli:

- studia e definisce il problema in oggetto;
- analizza le richieste e definisce quali sono i requisiti studiando i bisogni, siano essi impliciti o espliciti;
- analizza il fronte applicativo, in particolare gli utenti e i casi d'uso;
- redige lo *Studio di Fattibilità* e l'*Analisi dei Requisiti*.

Progettista

Il *Progettista* partendo dal lavoro svolto dall'*Analista*, ha il compito di sviluppare una soluzione che soddisfi i bisogni individuati. Tale compito ha natura sintetica, permettendo l'ottenimento di un'architettura che modelli il problema a partire da un insieme di requisiti. Chi ricopre questo ruolo:

- applica principi noti e collaudati per produrre un'architettura che assicuri coerenza e consistenza;
- deve produrre una soluzione sostenibile e realizzabile, che rientri nei costi stabiliti dal preventivo;
- si adopera per la costruzione di una struttura che soddisfi tutti i requisiti e che sia aperta alla comprensione;
- cerca di limitare il più possibile il grado di accoppiamento tra le varie componenti;
- rivolge il suo sforzo verso la ricerca dell'efficienza$_{\mathbf{G}}$, della flessibilità e della riusabilità;
- elabora una soluzione capace di interagire in modi diversi (per forma e per numero) con l'ambiente in cui si pone, e che sia sicura rispetto a eventuali anomalie e intrusioni esterne;
- ricerca la massima disponibilità ed affidabilità per l'architettura proposta.

Programmatore

Il *Programmatore* è la figura incaricata della Codifica. Egli deve implementare l'architettura prodotta dal *Progettista* in maniera che aderisca alle specifiche, ed è responsabile della manutenzione del codice creato. Egli:

- codifica secondo le specifiche stabilite dal *Progettista*. Il codice prodotto è documentato, versionabile ed è strutturato in maniera tale da agevolare la futura manutenzione;
- crea le componenti che serviranno poi per svolgere Verifica e Validazione del codice;
- scrive il *Manuale Utente* relativo al codice prodotto.

Verificatore

Il *Verificatore* è presente per tutta la durata del progetto. Egli si occupa di controllare che le attività svolte rispettino le norme e le attese prefissate. I compiti del *Verificatore* sono:

- controllare ed accertarsi che l'esecuzione delle attività di processo non abbia causato errori;
- redigere la parte retrospettiva del *Piano di Qualifica*, la quale descrive e chiarisce le verifiche e le prove effettuate.

4.1.5.5 Assegnazione dei compiti

La progressione nello svolgimento del progetto può essere vista come il completamento sequenziale o parallelo, di una serie di compiti. Ciascuno possiede una scadenza temporale e produce dei risultati utili alla realizzazione degli obiettivi posti.
I compiti sono determinati a volte dalla contingenza, a volte dai processi in atto, ma nella maggior parte dei casi sono dovuti ad entrambi i fattori. Per la loro assegnazione si è deciso di utilizzare il servizio di ticketing$_{\mathbf{G}}$ offerto da *GitHub*$_{\boldsymbol{G}}$, basato sul concetto di issue$_{\mathbf{G}}$.
Colui che si occupa della gestione dei compiti è il *Responsabile di Progetto* che deve:

- individuare il compito da svolgere;

- se ritiene il compito troppo complesso, lo può suddividere in più sotto-compiti;
- individuare uno o più componenti del gruppo al quale assegnare il compito (o i sotto-compiti) in base alle disponibilità e peculiarità di ciascuno e alla pianificazione presentata nel *Piano di Progetto 4.0.0-1.10*;
- aprire una issue, assegnando il compito al/ai soggetto/i definendone una data entro cui completarlo.

Di contro, i membri del gruppo:

- una volta che il compito viene loro assegnato, devono impegnarsi a svolgerlo entro i tempi stabiliti;
- completato il lavoro, sono responsabili della chiusura della issue di assegnazione del lavoro, dell'apertura di quella di verifica e infine di chiusura di quella di correzione.

Nella sezione sottostante viene trattato con maggior dettaglio quanto appena esposto.

4.1.5.6 Ciclo di vita del ticket

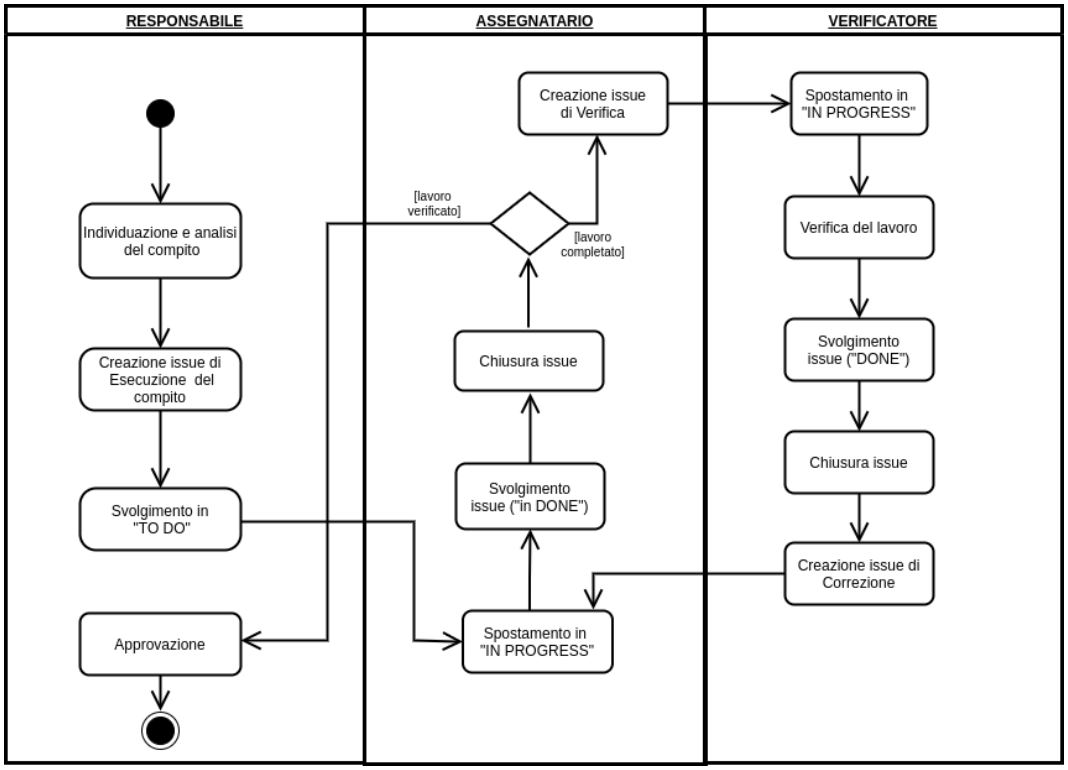

Figura 32: Ciclo di vita del ticket

Nel periodo di tempo che intercorre tra l'individuazione di un lavoro da svolgere e il suo completamento entrano in gioco le seguenti figure:

- ***Responsabile di Progetto***: individua un compito da svolgere, lo assegna ad un componente del gruppo con l'apertura della relativa issue; una volta che il lavoro è svolto e verificato, egli lo deve approvare;
- ***Assegnatario***: incaricato di svolgere il compito individuato, una volta terminato chiuderà la issue assegnatagli; successivamente, aprirà egli stesso una issue a carico del soggetto designato per la verifica;
- ***Verificatore***: persona deputata al controllo e alla verifica del lavoro svolto; concluso il suo compito, chiuderà la issue di verifica creandone un'altra per la correzione; essa conterrà le correzioni/modifiche da eseguire e sarà destinata all'*Assegnatario*.

Segue la descrizione dettagliata del ciclo di vita del ticket. Si vuole precisare che esso ha validità generale, indipendentemente dalla natura del compito da svolgere:

- **individuazione del compito**: il *Responsabile di Progetto* rileva determinate necessità, a cui bisogna rispondere con azioni opportune; traduce quanto identificato in richieste da soddisfare ed obiettivi da raggiungere, creando così un compito;
- **analisi del compito**: il *Responsabile di Progetto* stima la complessità del compito appena identificato, dividendolo in più sotto-compiti se lo ritiene troppo oneroso; individua poi uno o più *Assegnatari* e una data di scadenza entro la quale svolgere il lavoro;
- **creazione issue di esecuzione del compito**: il *Responsabile di Progetto* crea una issue utilizzando *GitHub*, definendo *Assegnatario* e data di scadenza stabiliti precedentemente;
- **spostamento issue in "TO-DO"**: una volta creata la issue, il *Responsabile di Progetto* la sposta nell'insieme "TO-DO", indicante tutti i compiti individuati, ma il cui svolgimento non è ancora cominciato;
- **spostamento issue in "IN PROGRESS"**: una volta che l'*Assegnatario* decide di iniziare il lavoro, sposta la issue nella categoria "IN PROGRESS";
- **svolgimento issue**: in questo periodo di tempo il compito viene portato a termine dal relativo *Assegnatario*;
- **spostamento issue in "DONE"**: l'*Assegnatario* sposta la issue nella categoria "DONE", per indicare che ha completato il suo lavoro;
- **chiusura della issue**: l'*Assegnatario* terminato il lavoro, chiude la sua issue e ne apre una di verifica da attribuire al *Verificatore*.
- **issue di verifica e correzione**: il *Verificatore* incaricato svolgerà la sua mansione di controllo, giudicando quanto fatto dall'*Assegnatario*; al termine, sarà compito del *Verificatore* stesso chiudere la issue di verifica, creandone poi un'altra per la correzione; esse si verranno a trovare nelle stesse situazioni ("TO-DO", "IN PROGRESS" e "DONE"), sopra esposte; una volta concluse tutte le correzioni da parte dell'*Assegnatario*, sarà quest'ultimo a provvedere alla chiusura della suddetta issue.
- **approvazione del lavoro**: il *Responsabile di Progetto* esamina per un'ultima volta il lavoro, controllando che rispetti gli obbiettivi fissati in partenza, e lo approva.

4.1.5.7 Gestione dei rischi

Per portare a termine il progetto vanno localizzati gli ostacoli che possono insorgere durante il suo sviluppo, a tal fine si necessita di un piano di gestione dei rischi.
È il *Responsabile di Progetto* incaricato di rilevare i rischi e renderli noti, documentandoli nel *Piano di Progetto 4.0.0-1.10*. Nel caso ne trovi di nuovi dovrà provvedere ad aggiungerli nell'analisi dei rischi. La procedura da seguire per la Gestione dei rischi rispetta quanto segue:

- individuazione di nuovi problemi e monitoraggio dei rischi già previsti;
- registrazione di ogni riscontro previsto dai rischi nel *Piano di Progetto 4.0.0-1.10*;
- aggiunta di nuovi rischi individuati nel *Piano di Progetto 4.0.0-1.10*;
- ridefinizione, se necessario, delle strategie di gestione dei rischi.

Codice identificativo

Ogni rischio viene strutturato come segue:

- **Codice**: compone la prima riga della tabella; viene fatto seguire da un "-" (trattino) e da un breve titolo esplicativo del contenuto della stessa (e.g: RT1 - Inesperienza tecnologica);
- **Descrizione**: concisa e sintetica presenta in poche righe il rischio riscontrato;
- **Conseguenze**: espone gli effetti a cui il rischio potrebbe portare al lavoro del gruppo e/o al progetto stesso, nel caso non venisse trattato adeguatamente;
- **Probabilità di occorrenza**: distinta fra Alta, Medio, Bassa rappresenta la probabilità che la situazione di rischio si manifesti;
- **Pericolosità**: come sopra viene distinta fra Alta, Medio, Bassa; la sua attribuzione permette di classificare il rischio in base alla pericolosità;
- **Precauzioni**: espone le strategie da attuare per cercare di evitare il verificarsi della situazione di rischio;
- **Piano di contingenza**: presenta il comportamento da adottare nel caso il rischio si verifichi effettivamente per cercare di limitarne il più possibile le conseguenze negative.

Per ciascun rischio è assegnato un codice identificativo articolato nella forma:

R[Tipologia][Numero]

dove

- **Tipologia:** identifica uno dei seguenti tipi di rischio:
  - **T**: rischi legati alle tecnologie;
  - **G**: rischi legati ai membri del gruppo;
  - **S**: rischi legati agli strumenti;
  - **O**: rischi legati all'organizzazione del lavoro;
  - **R**: rischi legati ai requisiti.
- **Numero**: incrementale a partire da 1; al cambio di tipologia riparte da 1.

Nel *Piano di Progetto 4.0.0-1.10* in §2 vengono riportati ed esposti i rischi individuati durante l'attività di progetto nei modi sopra descritti; per ogni rischio analizzato, in §A ne viene poi indicata l'attualizzazione rispetto al periodo d'insorgenza.

4.1.5.8 Metriche di processo

Per la valutazione dell'adeguatezza della pianificazione, il gruppo ha deciso di adottare le seguenti metriche$_{\mathbf{G}}$ di qualità di processo:

- **MPR02** Actual Cost of Work Performed (per una trattazione più estesa, consultare §D.1.2);
- **MPR03** Budgeted Cost of Work Scheduled (per una trattazione più estesa, consultare §D.1.3);
- **MPR04** Budgeted Cost of Work Performed (per una trattazione più estesa, consultare §D.1.4);
- **MPR05** Schedule Variance (per una trattazione più estesa, consultare §D.1.5);
- **MPR06** Budget Variance (per una trattazione più estesa, consultare §D.1.6).

4.2 Gestione dell'infrastruttura

4.2.1 Scopo

Il processo di Gestione dell'infrastruttura permette di stabilire e mantenere l'infrastruttura necessaria per qualsiasi altro processo. Essa può includere hardware, software, strumenti, tecniche, standard e strutture per lo sviluppo, il funzionamento o la manutenzione.

4.2.2 Descrizione

La presente sezione espone gli strumenti impiegati dal gruppo *ProApes* per quanto concerne le attività di Coordinamento e Pianificazione. Viene suddivisa per attività raggruppandone i corrispettivi strumenti; per ciascuno strumento viene fornita una breve descrizione annessa a link utili posti come note a piè di pagina.

4.2.3 Aspettative

Con il processo di Gestione dell'infrastruttura ci si aspetta di mantenere l'integrità di tutti i componenti dell'infrastruttura rendendoli accessibili a chi ne ha diritto.

4.2.4 Per il Coordinamento

Di seguito vengono trattati gli strumenti impiegati dal gruppo di lavoro per l'attività di Coordinamento.

Slack

Slack[6] è uno strumento collaborativo aziendale, utilizzato per inviare messaggi in modo istantaneo ai membri di un team di lavoro. L'applicativo permette ad un utente di iscriversi a diversi workspace$_{\mathbf{G}}$. Per ogni workspace di appartenenza è concessa la creazione di canali tematici dove sviluppare delle conversazioni specializzate. Ciò permette di evitare la trattazione di diversi argomenti su un'unica chat generale, come invece avviene in *Telegram*. *Slack* può inoltre venire integrato con diverse applicazioni. A tal proposito il gruppo ha provveduto ad integrarvi *GitHub* in modo che ogni azione svolta sulle repository del progetto venga sempre notificata da ogni membro del gruppo (che sia creazione di issue, commit,...). In tal modo si cerca promuovere l'incremento della collaborazione ed il tracciamento$_{\mathbf{G}}$ dei tasks eseguiti da ciascuno in un momento specifico.

Discord

Discord[7] è uno strumento gratuito di messaggistica istantanea e VoIP$_{\mathbf{G}}$. Nato nell'ambito dei videogiochi *Discord* è sostanzialmente un'app per chattare, che si può usare su smartphone, tablet e pc. Con *Discord* gli utenti possono:

- scriversi messaggi di testo;
- comunicare tra loro collegando il microfono oppure fare delle videochiamate a gruppi.

È quindi uno strumento che permette di comunicare in tempo reale, fare videochat di gruppo rimanendo allo stesso tempo un forum. Permette inoltre di creare dei sotto-canali dedicati classificabili per tema, in questo modo vi possono essere attive più conversazioni, spartite tra i vari membri del gruppo.
Il gruppo ha deciso di utilizzarlo per gestire la comunicazione interna per quanto concerne le videochiamate (come descritto in §4.1.4.4).

[6] https://slack.com/intl/en-it/
[7] https://discordapp.com/company/

Telegram

Telegram[8] è uno strumento di messaggistica istantanea, basato su cloud$_{\mathbf{G}}$, open-source che il gruppo *ProApes* ha impiegato durante le sue prime settimane di vita. Sebbene si sia deciso di utilizzare primariamente *Slack*, nel caso di imprevisti che rendano impossibile o ritardino la comunicazione, *Telegram* rimane comunque una valida alternativa. Alcune funzionalità di cui questo strumento dispone sono:

- possibilità di accedere da diversi dispositivi contemporaneamente, cosa attuabile grazie alla sincronizzazione istantanea del cloud;
- chat cloud con cifratura client-server; essa permette il salvataggio della conversazione cifrata sul server di *Telegram*;
- vasta gamma d'impiego grazie alla possibilità di inviare messaggi di testo, messaggi vocali, videomessaggi, posizione GPS attuale, coordinate GPS sulla mappa, contatti e file (con una dimensione massima di 1.5 GB);
- creazione di chat segrete; la loro conversazione non viene salvata sul server grazie ad una cifratura end-to-end$_{\mathbf{G}}$ rendendole visualizzabili solo nel device utilizzato in quel momento;
- si possono avere fino a 200 membri in un gruppo, che possono salire fino a 200 mila nel caso di supergruppo (limite che può essere aumentato). È inoltre possibile impostare amministratori con permessi selezionabili;
- un gruppo può essere reso pubblico settando un username, in questo modo si può leggerne i messaggi senza doversi per forza unire;
- i canali sono chat dove solo l'amministratore può inviare messaggi ai membri; un canale può contenere un numero illimitato di utenti e può essere sia pubblico che privato;
- presenza di bot, essi sono degli account *Telegram*, gestiti da un programma; offrono molteplici funzionalità attraverso risposte immediate e automatizzate.

Il team fa uso di un gruppo privato dove vi sono presenti tutti i membri e di Bot per creare sondaggi.

Gmail

Gmail[9] è uno strumento open-source di posta elettronica fornito da Google$_{\mathbf{G}}$. Il servizio gestisce la casella e-mail del gruppo, come descritto in §4.1.4.4. Nella stessa vengono integrati nativamente, anche altri strumenti come *Google Calendar*$_{\boldsymbol{G}}$.

Google Calendar

Scelto per la gestione degli eventi ed impegni del gruppo, *Google Calendar*[10] viene impiegato per indicare:

- incontri con proponente e committenti;
- scadenze che il gruppo deve rispettare durante lo svolgimento del progetto, in particolare:
  - scadenze interne fissate dai membri del team stesso per garantire un monitoraggio costante circa l'andamento del lavoro;
  - date delle Revisioni di avanzamento e delle baseline$_{\mathbf{G}}$ fissate dai committenti.
- disponibilità orarie di ogni membro del gruppo.

[8] https://telegram.org/
[9] https://www.google.com/intl/it/gmail/about/
[10] https://www.google.com/calendar

Google Drive

Google Drive$_{\mathbf{G}}$[11] è stato scelto per condividere velocemente file quali guide, documentazioni, riferimenti utili e documenti informali, facendosi supportare anche da *Google Docs*[12]. Quest'ultimo permette la creazione di documenti maneggevoli quando più soggetti vi lavorano in simultanea (ad esempio quando si devono stilate delle domande specifiche da fare a proponente/committenti). In questo *Google Docs* si rivela estremamente utile, visualizzando in tempo reale le modifiche apportate al documento e permettendo l'aggiunta di commenti.

Hangouts

Hangouts[13] è uno strumento di messaggistica istantanea e VoIP offerto come servizio di comunicazione da Google. Offre numerose integrazioni tramite piattaforme e plug-in$_{\mathbf{G}}$ compatibili con il browser di Google Chrome. La più comune è *Gmail*. *Hangouts* permette di effettuare chiamate e videochiamate, condividere foto esclusivamente con l'ausilio di una casella di posta *Gmail*, senza l'istallazione di applicazioni aggiuntive.
Il gruppo impiega questo strumento come alternativa per trattare con i soggetti esterni (come descritto in §4.1.4.4).

4.2.5 Per la Pianificazione

Di seguito vengono invece esposti gli strumenti utilizzati per l'attività di Pianificazione.

GanttProject

Per il supporto all'attività Pianificazione del progetto e alla realizzazione di diagrammi di Gantt$_{\mathbf{G}}$ all'unanimità si è deciso di utilizzare lo strumento *GanttProject*$_{\mathbf{G}}$[14], software open-source e multi-piattaforma.

4.3 Miglioramento

4.3.1 Scopo

Il processo di Miglioramento permette di stabilire, valutare, misurare, controllare e migliorare il ciclo di vita$_{\mathbf{G}}$ del software.

4.3.2 Descrizione

La presente sezione si compone nel modo che segue:

- istituzione del processo;
- valutazione del processo;
- miglioramento del processo.

4.3.3 Aspettative

Da questo processo ci si aspetta di riuscire ad operare nel pieno rispetto del miglioramento continuo, come esposto in §B.

[11] https://www.google.com/intl/it_ALL/drive/
[12] https://docs.google.com/
[13] https://hangouts.google.com/
[14] https://www.ganttproject.biz/

4.3.4 Istituzione del processo

Ciascun processo dovrà essere controllato, sviluppato e monitorato costantemente durante l'intero ciclo di vita del software; in tal modo si assicura il raggiungimento degli obiettivi fissati nella Pianificazione.

4.3.5 Valutazione del processo

In tale contesto verranno svolti i seguenti compiti:

- sviluppo, documentazione e applicazione di una procedura di valutazione del processo; tali valutazioni dovrebbero essere conservate e mantenute;
- pianificazione e svolgimento di revisioni dei processi ad intervalli appropriati per garantire la loro continua idoneità ed efficacia alla luce dei risultati della valutazione.

Per rispettare quanto sopra esposto il gruppo *ProApes* svolgerà alla fine di ogni periodo una procedura di valutazione per ciascun processo, nel rispetto della metodologia PDCA$_{\mathbf{G}}$. Nel *Piano di Qualifica 4.0.0-1.10* per ogni processo verranno inoltre riportate in forma tabellare le informazioni riguardanti:

- nome del processo;
- durata;
- tecnologie e strumenti impiegati;
- eventuali metriche di valutazione;
- problemi e ritardi;
- eventuali soluzioni e piani di contenimento;
- una valutazione quantificabile seguendo una scala da 1 (pessimo) a 10 (ottimale).

Per assicurare l'efficienza e l'efficacia dei processi verrà inoltre eseguita una loro revisione a intervalli appropriati.

4.3.6 Miglioramento del processo

Una volta completato il periodo:

- il team dovrà definire e applicare i miglioramenti necessari sorti dalla valutazione e della revisione del processo; la relativa documentazione dovrà essere aggiornata riflettendo effettivamente un miglioramento;
- dati storici, tecnici e di valutazione devono essere raccolti ed analizzati nel *Piano di Qualifica* per capire i punti deboli e le potenzialità dei processi impiegati. L'analisi deve essere usata come feedback per migliorare i processi, raccomandando modifiche da applicare alle procedure, agli strumenti e alle tecnologie.

4.3.7 Metriche

Per la valutazione della capacità di attuare il miglioramento, il gruppo ha deciso di adottare lo standard$_{\mathbf{G}}$ *SPICE*$_{\mathbf{G}}$ come metrica di qualità di processo; tale standard sarà applicato a tutti i processi istanziati nella fase corrente (per una trattazione più estesa, consultare §D.1.1);

4.4 Formazione del personale

4.4.1 Scopo

Il processo di Formazione permette di avere e mantenere nel tempo personale competente. La Fornitura, lo Sviluppo, la Manutenzione ecc. dei prodotti software, dipendono infatti in gran parte dalla presenza di personale idoneo e preparato. È pertanto imperativo che tale processo sia pianificato e implementato in anticipo in modo da avere sempre a disposizione personale istruito.

4.4.2 Descrizione

Ciascun membro del gruppo deve provvedere alla propria formazione autonomamente, studiando le tecnologie usate e colmando eventuali carenze. In particolare, i soggetti in deficit dovranno colmare le loro mancanze approfondendo lo studio personale; sarà compito dei membri più esperti condividere le conoscenze per operare secondo il principio del miglioramento continuo.
Il team farà riferimento alla documentazione di seguito esposta, oltre a quella reperita per proprio conto.

4.4.3 Aspettative

Ciò che ci si attende da tale processo è di poter assicurare una qualità del lavoro che rispetti le aspettative, grazie alla presenza di personale esperto e competente in grado di svolgere i compiti assegnatogli.

4.4.3.1 Creazione di documenti

- LaTeX: https://www.latex-project.org;
- *TexStudio*: https://www.texstudio.org;
- *Aspell*: http://aspell.net/.

4.4.3.2 *Node.js* e tecnologie correlate

- *NodeJS*$_G$: https://nodejs.org/en/docs/;
- *Angular*$_G$: https://angular.io/;
- *Angular/CLI*$_G$: https://cli.angular.io/;
- *Typescript*$_G$: https://www.typescriptlang.org/;
- *HTML5*$_G$: https://www.w3schools.com/html/html5_intro.asp;
- *CSS3*$_G$: https://www.w3schools.com/css/;
- *Npm*$_G$: https://www.npmjs.com/;
- *ReactJS*$_G$: https://grafana.com/blog/2019/03/26/writing-react-plugins/.

4.4.3.3 Servizi di terze parti

- *GitHub*$_G$: https://help.github.com;
- *Git*$_G$: https://git-scm.com;
- *Grafana*$_G$: https://grafana.com/docs/grafana/latest/;
- *VirtualBox*$_G$: https://www.virtualbox.org/wiki/Documentation.

4.4.3.4 Test, Code Coverage$_G$ e Continuous Integration$_G$

- **test d'unità$_G$**:
  http://softwaretestingfundamentals.com/unit-testing/;
- **test d'integrazione$_G$**:
  http://softwaretestingfundamentals.com/integration-testing/;
- **test d'accettazione$_G$**:
  http://softwaretestingfundamentals.com/acceptance-testing/;
- **test di sistema$_G$**:
  http://softwaretestingfundamentals.com/system-testing/;
- ***Jest$_G$***: https://jestjs.io;
- ***GitHub Action$_G$***: https://help.github.com/en/actions.

A Standard ISO/IEC 15504 - SPICE

ISO/IEC 15504, noto anche come *Software Process Improvement and Capability Determination (SPICE)* è un insieme di standard$_{\mathbf{G}}$ tecnici per i processi di sviluppo del software. ISO/IEC 15504 è stato rivisto e corretto in un nuovo standard, l'ISO/IEC 33001 nel 2015. Tuttavia, resta un valido riferimento per quanto concerne la valutazione della qualità di processi software; rispetto ad esso possono essere misurati gli esiti delle prove di qualità, per poi ricavarne un quadro complessivo delle capacità di organizzazione per il rilascio$_{\mathbf{G}}$ dei prodotti software.

A.1 Modello di riferimento

Il modello di riferimento per *SPICE* definisce una *dimensione di processo* e un *livello di capacità*.

A.1.1 Classificazione dei processi

ISO/IEC 15504 considera i seguenti tipi di processo$_{\mathbf{G}}$:

- **cliente-fornitore**;
- **ingegneristico**;
- **di supporto**;
- **gestionali**;
- **organizzativi**.

A.1.2 Livelli di *capability*

A.1.2.1 Il concetto di *capability*

Nell'acronimo *SPICE*, il termine *Capability* è la capacità di un processo di essere cognitivamente capace di raggiungere il suo scopo. Un processo con un'alta *capability* è attuato da tutto il team di sviluppo in modo disciplinato e sistematico; la bassa *capability* di un processo indica una sua istanziazione poco professionale da parte del team.

A.1.2.2 Livelli

La *capability* di un processo viene misurata su una scala di sei livelli, a cui sono assegnati degli attributi di valutazione della qualità del processo.
I livelli considerati sono i seguenti:

- **livello 0 *incomplete***: il processo di livello 0 non è implementato né in grado di raggiungere i propri obiettivi; non viene prodotto alcun output significativo.
  Non ci sono attributi associati a questo livello.
- **livello 1 *performed***: il processo di livello 1 è implementato e raggiunge gli obiettivi prestabiliti. Tuttavia non è sottoposto a controlli costanti ai fini di correzione e miglioramento; gli output che produce sono identificabili.
  L'attributo associato al livello è:
  - **performance di processo (PP)**: riporta il numero di obiettivi raggiunti.
- **livello 2 *managed***: il processo di livello 2 è gestito tramite pianificazione, controllo e correzione. L'output risultante, tracciato e controllato, raggiunge gli obiettivi fissati.
  Attributi associati al livello sono:
  - **gestione della performance (PM)**: riporta il grado di organizzazione degli obiettivi fissati;

  - **gestione del prodotto di lavoro (WPM)**: riporta il grado di organizzazione dei prodotti rilasciati.

- **livello 3 *established***: un processo di livello 3 è definito a partire da uno standard e quindi regolamentato da principi dettati dall'Ingegneria del Software. L'output risultante impiega una quantità di risorse limitata.
  Attributi associati al livello sono:

  - **definizione di processo (PDEF)**: riporta il grado di adesione del processo agli standard;
  - **rilascio di processo (PDEP)**: riporta in che misura il processo può essere rilasciato con garanzia di ripetibilità$_{\mathbf{G}}$.

- **livello 4 *predictable***: un processo livello 4 è istanziato in modo coerente entro limiti definiti; è caratterizzato dalla raccolta e dal monitoraggio di misure dettagliate, funzionali al consolidamento della prevedibilità del processo.
  Attributi associati al livello sono:

  - **misurazioni di processo (PME)**: riporta con quanta efficacia$_{\mathbf{G}}$ le metriche$_{\mathbf{G}}$ possono essere applicate al processo;
  - **controllo di processo (PC)**: riporta il grado di predicibilità dei risultati delle valutazioni.

- **livello 5 *optimizing***: un processo di livello 5 è correttamente definito e tracciato, soggetto a continui analisi e miglioramento.
  Attributi associati al livello sono:

  - **innovazione di processo (PI)**: riporta quanto i cambiamenti attuati nel processo risultano innovativi e positivi, mediante una fase di analisi;
  - **ottimizzazione di processo (PO)**: riporta quanto la curva di miglioramento del processo sia lineare, a rappresentazione di una corretta gestione del rapporto tra risorse impiegate e risultati ottenuti.

Ogni attributo è valutato con un giudizio su una scala di quattro livelli, a ciascuno dei quali corrisponde una classe di valori percentuali che esprimono numericamente il grado di soddisfacimento dell'attributo:

- **N *not achieved***: 0% - 15%; il processo non ha implementato l'attributo o presenta gravi lacune in merito;
- **P *partially achieved***: >15% - 50%; il processo ha implementato l'attributo in modo sistematico, ma risulta ancora migliorabile e poco predicibile;
- **L *largely achieved***: >50% - 85%; il processo ha ampiamente implementato l'attributo in modo sistematico, ma il suo valore risulta poco uniforme all'interno delle varie parti del processo;
- **F *fully achieved***: >85% - 100%; il processo ha implementato completamente l'attributo in modo sistematico e uniforme in ogni sua parte.

B PDCA - Ciclo di Deming

Figura 33: PDCA, Fonte: iSigma group

Il ciclo PDCA (noto anche come ciclo di Deming) è un metodo iterativo diviso in quattro fasi utilizzato per il controllo e il miglioramento continuo di processi e prodotti.
Le fasi vengono strutturate in questo modo:

- **Plan**: si stabiliscono gli obbiettivi e i processi necessari per raggiungere i risultati in accordo con quanto atteso. Si pianifica nei minimi dettagli, tuttavia è utile predisporre miglioramenti di dimensione ridotta in quanto monitorabili più facilmente;
- **Do**: si esegue ciò che è stato pianificato; oltre ad applicare le decisioni prese è necessario raccogliere dati che verranno analizzati nelle fasi successive;
- **Check**: si confrontano i dati raccolti durante la fase di *Do* con quanto era stato previsto durante la fase di *Plan*. Risulta così possibile valutarne l'esito positivo e capire se i nuovi processi portano valore aggiunto o se possono essere migliorati ulteriormente;
- **Act**: fase di miglioramento del processo$_{\mathbf{G}}$. Si consolidano gli aspetti positivi mentre le differenze significative riscontrate nella fase di *Check* vengono analizzate e corrette. Al termine di questa fase si chiude il ciclo; le attività pianificate nella fase corrente concorrono a creare una nuova fase di plan.

C Standard ISO/IEC 9126

ISO/IEC 9126 è uno standard$_G$ internazionale per la valutazione della qualità software. Attualmente è stato sostituito dal ISO/IEC 25010:201, tuttavia è ancora un caposaldo in materia di qualità, motivo per cui il gruppo *ProApes* ha deciso di adottarlo come standard di riferimento.
Questo standard permette di diffondere una comprensione comune degli obiettivi di un progetto software.

C.1 Modello della qualità del software

Il modello di qualità stabilito dallo standard si articola in sei caratteristiche principali, ognuna delle quali a sua volta presenta delle sottocaratteristiche, misurabili tramite delle metriche$_G$ di qualità.
Di seguito sono esposte le sei caratteristiche sopra citate:

- **Funzionalità**: insieme di attributi riguardanti un insieme di funzioni e le loro proprietà. Tali funzioni mirano a soddisfare requisiti stabiliti o implicitamente dedotti.
  Le sottocaratteristiche della *funzionalità* sono:
  - **Appropriatezza**: capacità del prodotto di fornire un insieme di funzioni in grado di svolgere compiti e soddisfare obiettivi prefissati;
  - **Accuratezza**: capacità del prodotto di fornire i risultati desiderati con la precisione richiesta;
  - **Interoperabilità**: capacità del prodotto di interagire ed operare con uno o più sistemi;
  - **Sicurezza**: capacità del prodotto di proteggere informazioni e dati monitorando gli accessi ad essi;
  - **Aderenza alla funzionalità**: grado di adesione del prodotto agli standard scelti dal team, alle convenzioni e ai regolamenti;
- **Affidabilità**: insieme di attributi riguardanti la capacità del prodotto di mantenere un dato livello di performance sotto condizioni di esecuzione prestabilite.
  Le sottocaratteristiche dell'*affidabilità* sono:
  - **Maturità**: capacità del prodotto di evitare errori e risultati non corretti durante l'esecuzione;
  - **Tolleranza agli errori**: capacità del prodotto di conservare il livello di prestazioni anche in caso di malfunzionamenti o di uso inappropriato del prodotto;
  - **Recuperabilità**: capacità di un prodotto di ripristinare il livello di prestazioni e di recupero delle informazioni rilevanti, a seguito di un malfunzionamento. Il periodo di inaccessibilità del prodotto a seguito di un errore è valutato proprio dalla recuperabilità;
  - **Aderenza all'affidabilità**: grado di adesione del prodotto a standard, regole e convenzioni inerenti all'affidabilità.
- **Efficienza**: insieme di attributi riguardanti il rapporto tra il livello delle prestazioni e la quantità di risorse usate durante la loro esecuzione, sotto condizioni prestabilite.
  Le sottocaratteristiche dell'*efficienza* sono:
  - **Comportamento rispetto al tempo**: capacità di un prodotto di fornire appropriati tempi di risposta e di elaborazione e quantità di lavoro eseguendo le funzionalità richieste in date condizioni di lavoro;
  - **Utilizzo delle risorse**: capacità di un prodotto di usare appropriati numero e tipo di risorse durante la fase di esecuzione sotto condizioni di utilizzo date;

  - **Aderenza all'efficienza**: grado di adesione del prodotto a standard, regole e convenzioni inerenti all'efficienza.

- **Usabilità**: insieme di attributi riguardanti lo sforzo necessario all'utilizzo del prodotto e la valutazione individuale su tale uso da parte di un insieme di utenti.
  Le sottocaratteristiche dell'*usabilità* sono:
  - **Comprensibilità**: facilità di comprensione dei concetti base del prodotto, per permettere all'utente di capire se il prodotto è appropriato;
  - **Apprendibilità**: facilità con cui un utente medio può comprendere il funzionamento del prodotto ed imparare ad usarlo;
  - **Operabilità**: misura della possibilità degli utenti di usare il prodotto in vari contesti e di adattarlo ai propri scopi;
  - **Attrattiva**: misura della gradevolezza e dell'essere "attraente" del prodotto durante l'uso;
  - **Aderenza all'usabilità**: grado di adesione del prodotto a standard, regole e convenzioni inerenti all'usabilità
- **Manutenibilità**: insieme di attributi riguardanti lo sforzo richiesto per apportare modifiche specifiche al prodotto.
  Le sottocaratteristiche della *manutenibilità* sono:
  - **Analizzabilità**: misura della difficoltà incontrata nel diagnosticare un errore nel prodotto;
  - **Modificabilità**: facilità di apportare modifiche al prodotto originale o di introdurre nuove funzionalità; per modifiche si intendono cambiamenti al codice, alla progettazione o alla documentazione;
  - **Stabilità**: capacità del prodotto di evitare effetti indesiderati a causa di modifiche;
  - **Testabilità**: capacità del prodotto di essere verificato;
  - **Aderenza alla manutenibilità**: grado di adesione del prodotto a standard, regole e convenzioni inerenti all'usabilità.
- **Portabilità**: insieme di attributi riguardanti la capacità del software di essere trasferito da un ambiente di esecuzione ad un altro.
  Le sottocaratteristiche della *portabilità* sono:
  - **Adattabilità**: capacità del prodotto di essere adattato per differenti ambienti operativi senza richiedere azioni specifiche diverse da quelle previste dal prodotto per quell'attività; include la scalabilità delle capacità interne del prodotto;
  - **Installabilità**: capacità del prodotto di essere installato in un dato ambiente;
  - **Coesistenza**: capacità di un prodotto di coesistere con altre applicazioni in ambienti comuni e condividere le risorse;
  - **Aderenza alla portabilità**: capacità del prodotto di aderire a standard e convenzioni relative alla portabilità.

C.2 Qualità esterna

C.2.1 Metriche per la qualità esterna

Le metriche esterne, specificate nella sezione *ISO/IEC 9126-2* dello standard, misurano i comportamenti del prodotto sulla base dei test, dall'operatività e dall'osservazione durante la sua esecuzione, in funzione degli obiettivi stabiliti.

C.3 Qualità interna

C.3.1 Metriche per la qualità interna

Le metriche interne, specificate nella sezione *ISO/IEC 9126-3* si applicano al prodotto non eseguibile (e.g: al codice sorgente) durante la progettazione e la codifica. Sono anche dette *misure statiche*.
Le misure effettuate permettono di prevedere il livello di qualità esterna ed in uso del prodotto finale, poiché gli attributi interni influiscono su quelli esterni e quelli in uso.
Le metriche interne permettono di individuare eventuali problemi che potrebbero inficiare la qualità finale del prodotto prima che sia realizzato il prodotto eseguibile.

C.4 Qualità d'uso

La qualità in uso rappresenta il punto di vista dell'utente sul prodotto. Il livello di qualità in uso è raggiunto quando sono state conseguite sia la qualità esterna (dinamica) che quella interna (statica).

C.4.1 Metriche per la qualità d'uso

Le norme *ISO/IEC 9126-4* forniscono esempi di metriche da utilizzare per la misurazione della qualità rispetto ai tre punti di vista: interno, esterno, in uso.
La qualità in uso permette di abilitare specificati utenti ad ottenere dati obiettivi con efficacia, produttività, sicurezza e soddisfazione.

- **Efficacia**: capacità del prodotto di consentire agli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;
- **Produttività**: capacità di consentire agli utenti di adoperare un'appropriata quantità di risorse rispetto all'efficacia ottenuta in uno dato contesto d'uso;
- **Soddisfazione**: capacità del prodotto di corrispondere alle aspettative degli utenti;
- **Sicurezza**: capacità del prodotto di raggiungere accettabili livelli di rischio di danni a persone, software, apparecchiature tecniche e all'ambiente d'uso.

D Metriche di qualità

D.1 Metriche per la qualità del processo

La presente sezione espone le metriche selezionate dal gruppo per misurare il raggiungimento degli obiettivi di qualità del processo.

D.1.1 MPR01 SPICE

Utilizzata al termine di ogni periodo per monitorare e valutare la qualità dei processi impiegati; tale standard viene illustrato nel dettaglio in §A.

D.1.2 MPR02 *Actual Cost of Work Performed*

Valore intero che corrisponde alla somma di tutte le spese sostenute dal gruppo fino al momento in cui l'indice viene calcolato; il limite superiore al valore di questo indice è il preventivo presentato nel *Piano di Progetto 4.0.0-1.10*.

D.1.3 MPR03 *Budgeted Cost of Work Scheduled*

Valore intero che corrisponde alla spesa pianificata per l'attività di progetto alla data corrente; questo valore è reperibile nella sezione del *Piano di Progetto 4.0.0-1.10.* dedicata al preventivo.

D.1.4 MPR04 *Budgeted Cost of Work Performed*

Valore intero che corrisponde al valore effettivo del prodotto ottenuto fino al momento in cui l'indice viene calcolato.

D.1.5 MPR05 *Schedule Variance*

Indice temporale che riporta se il gruppo è in linea con la programmazione predisposta, o di quanto se ne discosta.
La formula adottata è

$$SV = BCWP - BCWS$$

dove

- **SV** è la sigla per *Scheduled Variance*;
- **BCWP** è la sigla per *Budgeted Cost of Work Performed*, valore effettivo alla data corrente;
- **BCWS** è la sigla per *Budgeted Cost of Work Scheduled*, valore programmato per la data corrente.

In base al risultato ottenuto:

- **SV** $>$ **0** indica che il gruppo è in anticipo rispetto alla pianificazione; in futuro, le previsioni dovranno essere eseguite con più precisione, tenendo conto di questo risultato;
- **SV** $=$ **0** indica che il gruppo è in linea con la pianificazione; i criteri adottati per fare la pianificazione sono quindi efficaci, e dovranno essere usati anche per previsioni future;
- **SV** $<$ **0** indica che il gruppo è in ritardo rispetto alla pianificazione: è necessaria una revisione delle pianificazioni da quel momento in poi, in modo da ridistribuire le risorse rimaste ed evitare di accumulare ulteriori ritardi.

D.1.6 MPR06 *Budget Variance*

Indice di costo che segnala se il gruppo rientra nel budget di spesa previsto per la data corrente.
La formula adottata è

$$BV = BCWS - ACWP$$

dove

- **BV** sta per *Budget Variance*;
- **BCWS** (*Budgeted Cost of Work Scheduled*) indica il costo pianificato per la realizzazione delle attività alla data corrente;
- **ACWP** (*Actual Cost of Work Performed*) indica il costo effettivamente sostenuto per la realizzazione delle attività alla data corrente.

Una *Budget Variance* positiva segnala che le spese rientrano nei limiti previsti; se fosse negativa segnalerebbe una spesa eccessiva alla data corrente, che potrebbe portare, sul lungo termine, a violare i termini del contratto coi committenti riguardanti il budget.

D.2 Metriche per la qualità del prodotto

La presente sezione espone le metriche selezionate dal gruppo per misurare il raggiungimento degli obiettivi di qualità del prodotto.

D.2.1 MPDD01 Indice di *Gulpease*

Indice che riporta il grado di leggibilità di un testo redatto in lingua italiana.
La formula adottata è

$$GULP = 89 + \frac{300 * (totale\ frasi) - 10 * (totale\ lettere)}{totale\ parole}$$

L'indice così calcolato può assumere valori tra 0 e 100:

- **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.

D.2.2 MPDD02 Errori ortografici

Il controllo ortografico viene eseguito secondo quanto descritto in §3.1.11 del presente documento.

D.2.3 MPDS01 Copertura requisiti obbligatori

Indice che misura in ogni istante la percentuale di requisiti obbligatori soddisfatti.
La formula adottata è

$$RO = (\frac{ROC}{RO}) * 100$$

dove

- **ROC** indica il numero di requisiti obbligatori coperti dall'implementazione;
- **RO** indica il numero complessivo di requisiti obbligatori.

D.2.4 MPDS02 Copertura requisiti accettati

Percentuale di requisiti desiderabili e facoltativi accettati effettivamente soddisfatti.
La formula adottata è

$$CRA = (\frac{RAC}{RA}) * 100$$

dove

- **RAC** indica il numero di requisiti accettati coperti dall'implementazione;
- **RA** indica il numero complessivo di requisiti accettati.

D.2.5 MPDS03 Numero di *bug*

Numero di righe di codice che potrebbero comportare un comportamento imprevisto o diverso da quello desiderato.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.6 MPDS04 Numero di *code smell*

Numero di *code smell* individuati, corrispondenti a righe di codice difficilmente estendibili.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.7 MPDS05 *Technical Debt*

Lavoro necessario a sanare *code smell*. Il valore viene misurato in ore di lavoro.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.8 MPDS06 *Remediation effort*

Lavoro necessario a risolvere *bug*. Il valore viene misurato in ore di lavoro.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.9 MPDS07 Complessità ciclomatica

Indice software utilizzato per stimare la complessità di un programma attraverso l'analisi del codice sorgente. Viene calcolata impiegando il *grafo del flusso di controllo*, dove:

- i nodi sono gruppi indivisibili di istruzioni;
- un arco connette due nodi se le istruzioni di un nodo possono essere eseguite subito dopo le istruzioni dell'altro.

Un valore troppo elevato di tale metrica è il risultato di codice troppo complesso, difficilmente testabile e manutenibile.
Esistono due metodologie per calcolare la complessità ciclomatica; il primo adotta la formula

$$V(G) = E - N + 2P$$

dove

- **E** è il numero di archi;
- **N** è il numero di nodi;
- **P** è il numero di componenti connesse[15].

La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

[15] Per una singola unità di programma P il valore è pari a 1

D.2.10 MPDS08 Complessità cognitiva

Misura della difficoltà di comprensione del codice.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.11 MPDS09 Successo dei test

Percentuale di test implementati e superati correttamente.
La formula adottata è

$$PSTE = (\frac{TSUP}{TTOT}) * 100$$

dove

- **TSUP** indica il numero di test superati;
- **TOTT** indica il numero complessivo di test implementati.

La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.12 MPDS10 *Line coverage*

Percentuale di linee di codice verificate da test automatici.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.13 MPDS11 *Branch coverage*

Percentuale del rapporto tra tutte le possibili deviazioni del flusso del codice (costrutti `if else`, operatori ternari, lancio di eccezioni e altre istruzioni similari).

La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.14 MPDS12 *Code coverage*

Percentuale di copertura del codice sorgente da parte dei test. Si ricava da diversi fattori di coverage secondo la formula[16]:

$$coverage = (CT + CF + LC)/(2 * B + EL)$$

dove:

- **CT** indica i branch valutati positivi almeno una volta;
- **CF** indica i branch valutati negativi almeno una volta;
- **LC** indica le righe di codice coperte da test;
- **B** indica il numero totale di branch;
- **EL** indica il numero totale di righe di codice da coprire coi test.

La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.15 MPDS13 Densità di duplicazione

Percentuale di righe di codice ripetute tra i vari file sorgente. Un alto valore di questo indice significa una bassa qualità del codice prodotto e una sua bassa ottimizzazione.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

[16] https://docs.sonarqube.org/latest/user-guide/metric-definitions/#Metricdefinitions-Tests

D.2.16 MPDS14 Rapporto fra linee di commento e linee di codice

Rapporto tra linee di commento e linee di codice vero e proprio.
La formula adottata è

$$RCC = (\frac{LCOM}{LC}) * 100$$

dove

- **LCOM** indica il numero di linee di commento;
- **LC** indica il numero totale di linee di codice presenti.

La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.

D.2.17 MPDS15 Numero di nuove righe

Numero di nuove righe di codice introdotte all'ultimo `commit`.
La metrica viene rilevata automaticamente mediante l'utilizzo di *Sonarcloud*$_{\mathbf{G}}$.