Norme di Progetto

7DOS - 22 Marzo 2019

Informazioni sul documento	
Versione	3.0.0
Responsabile	Giacomo Barzon
Verifica	Giovanni Sorice Nicolò Tartaggia
Redazione	Lorenzo Busin Andrea Trevisin Michele Roverato
Stato	Approvato
Uso	Interno
Destinato a	Prof. Tullio Vardanega Prof. Riccardo Cardin 7DOS
Email	7dos.swe@gmail.com

Descrizione

Questo documento descrive le regole, gli strumenti e le convenzioni adottate durante la realizzazione del progetto *G&B*.

Diario delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
3.0.0	2019-03-22	*Approvazione del documento per rilascio RQ*	Giacomo Barzon	Responsabile
2.6.0	2019-03-21	*Verifica del documento*	Giovanni Sorice	Verificatore
2.5.0	2019-03-20	*Verifica del documento*	Nicolò Tartaggia	Verificatore
2.4.0	2019-03-19	*Metriche associate alle attività a cui riferiscono, per una maggiore coesione informativa - (vedi Verbale del 2019-03-18)*	Lorenzo Busin	Amministratore
2.3.0	2019-03-19	*Ampliamento §4.1 e §4.2*	Andrea Trevisin	Amministratore
2.2.4	2019-03-19	*Aggiunte figure in §2.2.2*	Lorenzo Busin	Amministratore
2.2.3	2019-03-19- (vedi Verbale del 2019-03-18)	*Ampliamento §2.1*	Lorenzo Busin	Amministratore
2.2.1	2019-03-19	*Modificato §B - (vedi Verbale del 2019-03-18)*	Michele Roverato	Amministratore
2.2.0	2019-03-18	*Ampliamento §3 - (vedi Verbale del 2019-03-18)*	Michele Roverato	Amministratore
2.1.1	2019-03-18	*Riferimenti a standard spostati in §1.4.2 - (vedi Verbale del 2019-03-18)*	Andrea Trevisin	Amministratore
2.1.0	2019-03-18	*Stesura §3.4 e §3.3*	Andrea Trevisin	Amministratore
2.0.1	2019-03-18	*Spostate norme sul versionamento in §3 - (vedi Verbale del 2019-03-18)*	Michele Roverato	Amministratore
2.0.0	2019-02-07	*Approvazione del documento per rilascio RP*	Marco Costantino	Responsabile
1.6.0	2019-02-07	*Verifica del documento*	Giovanni Sorice	Verificatore
1.5.0	2019-02-05	*Verifica del documento*	Andrea Trevisin	Verificatore

Versione	Data	Descrizione	Autore	Ruolo
1.4.0	2019-02-04	*Ampliamento §2*	Michele Roverato	Amministratore
1.3.1	2019-02-04	*Fine stesura §D*	Giacomo Barzon	Amministratore
1.3.0	2019-02-03	*Inizio stesura §D*	Giacomo Barzon	Amministratore
1.2.1	2019-02-02	*Stesura §A, §B e §C*	Michele Roverato	Amministratore
1.2.0	2019-01-26	*Verifica del documento*	Andrea Trevisin	Verificatore
1.1.0	2019-01-25	*Verifica del documento*	Giovanni Sorice	Verificatore
1.0.2	2019-01-23	*Correzioni §2.2.1.5, §3.1.10 e §4.4.4*	Lorenzo Busin	Amministratore
1.0.1	2019-01-21	*Correzioni §2*	Nicolò Tartaggia	Amministratore
1.0.0	2018-12-04	*Approvazione del documento per rilascio RR*	Nicolò Tartaggia	Responsabile
0.9.0	2018-12-04	*Verifica del documento*	Andrea Trevisin	Verificatore
0.8.0	2018-12-02	*Verifica del documento*	Giacomo Barzon	Verificatore
0.7.0	2018-12-01	*Fine stesura §4*	Marco Costantino	Amministratore
0.6.0	2018-12-01	*Fine stesura §3*	Lorenzo Busin	Amministratore
0.5.0	2018-12-01	*Fine stesura §2*	Giovanni Sorice	Amministratore
0.4.3	2018-11-29	*Sviluppo stesura §2*	Giovanni Sorice	Amministratore
0.4.2	2018-11-28	*Sviluppo stesura §4*	Michele Roverato	Amministratore
0.4.1	2018-11-27	*Sviluppo stesura §3*	Lorenzo Busin	Amministratore
0.3.0	2018-11-26	*Inizio stesura §3*	Lorenzo Busin	Amministratore
0.2.0	2018-11-26	*Inizio stesura §4*	Marco Costantino	Amministratore
0.1.0	2018-11-26	*Inizio stesura §2*	Giovanni Sorice	Amministratore
0.0.2	2018-11-26	*Stesura §1*	Giovanni Sorice	Amministratore

Versione	Data	Descrizione	Autore	Ruolo
0.0.1	2018-11-26	*Stesura dello scheletro del documento*	Lorenzo Busin	Amministratore

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il presente documento descrive e fissa tutte le norme, le convenzioni e gli strumenti che verranno adottati dal nostro team per assicurare un modus operandi comune a tutti i membri nello sviluppo del *progetto*$_g$. Questo suppone che tutti i componenti del gruppo abbiano preso visione del documento e ne abbiano concordato e accettato i modi per garantire la massima omogeneità e collaborazione per tutto il progetto.

1.2 Glossario

Per rendere la lettura del documento più semplice, chiara e comprensibile viene allegato il documento *Glossario v3.0.0* nel quale sono contenute le definizioni dei termini tecnici, dei vocaboli ambigui, degli acronimi e delle abbreviazioni. La presenza di un termine all'interno del Glossario è segnalata con una "g" posta come pedice (esempio: *Glossario*$_g$).

1.3 Maturità del documento

Il presente documento potrebbe essere soggetto ad incrementi futuri. Per questo motivo, non si pone l'obiettivo di risultare completo. Tutto ciò che riguarda la pianificazione degli incrementi, può essere trovato nel *Piano di Progetto v3.0.0* in §4.

1.4 Riferimenti

1.4.1 Normativi

- **Verbali:** *Verbale del 2018-12-04 e del 2019-03-18.*

1.4.2 Informativi

- **ISO/IEC 12207**:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- **ISO/IEC 29119**:
  https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-1:v1:en;
- ***JavaScript Style Guide and Coding Conventions:***$_g$
  https://www.w3schools.com/js/js_conventions.asp;
- ***Grafana***$_g$ **Code Styleguide:**
  http://docs.grafana.org/plugins/developing/code-styleguide/;
- ***Angular TypeScript***$_g$ **Code Styleguide:**
  https://angular.io/guide/styleguide;
- ***HTML5 Style Guide and Coding Conventions***$_g$**:**
  https://www.w3schools.com/html/html5_syntax.asp;

- ***Airbnb CSS / Sass Styleguide*$_g$:**
  https://github.com/airbnb/css;
- **Trender**:
  https://github.com/campagna91/Trender/tree/master;
- **Software Engineering - Ian Sommerville - 10th Edition**(Capitolo 2).

2 Processi primari

2.1 Fornitura

Il fine di questa sezione è definire le norme che i membri del gruppo 7DOS sono invitati a rispettare con l'obiettivo di proporsi e diventare fornitori nei confronti dell'azienda proponente Zucchetti s.r.l. e dei committenti Prof. Tullio Vardanega e Prof. Riccardo Cardin per quanto concerne il prodotto *G&B*. Per raggiungere questa meta nel miglior modo possibile, abbiamo intenzione di collaborare in modo *efficiente*$_g$ e *efficace*$_g$ con i referenti dell'azienda. I punti fondamentali che verranno affrontati insieme al proponente saranno:

- Determinare gli aspetti cruciali al fine di soddisfare l'azienda proponente;
- Concordare la qualifica del prodotto;
- Determinare vincoli sui processi e sui requisiti;
- Stimare i costi del prodotto finale.

Di seguito sono riportate tutte le attività che compongono questo processo:

2.1.1 Studio del dominio tecnologico

Consiste nella ricerca di tutte quelle informazioni relative alle tecnologie e nella valutazione di possibili alternative utili al fine del progetto. Ne consegue che i risultati ottenuti, frutto di informazioni reperite ed esperienze personali, devono prima essere discussi, approvati ed eventualmente concordati con la proponente. Ogni componente del gruppo è incaricato di approfondire o studiare autonomamente le tecnologie necessarie allo sviluppo del progetto, in modo da ottenere buona padronanza del dominio tecnologico impiegato.

2.1.2 Normazione

La definizione delle norme potrebbe subire delle variazioni con il progredire del progetto; ciò è dovuto a nuove necessità a cui il gruppo andrà incontro ed eventuali problematiche che potrebbero essere riscontrate. Sarà quindi *compito*$_g$ degli *Amministratori* sopperire a queste complicazioni redigendo nuove norme o modificando quelle già presenti.

2.1.3 Studio di Fattibilità

Nel documento *Studio di Fattibilità v1.0.0.* troviamo le motivazioni che hanno portato il nostro gruppo a favorire la scelta del prodotto per cui proporci come fornitori. Inoltre, si riportano, per ogni capitolato, le seguenti informazioni:

- **Descrizione**: riporta una breve sintesi del prodotto da sviluppare;
- **Studio del dominio**: riporta un'analisi del dominio applicativo, in cui vi è una più corposa descrizione del prodotto da sviluppare con l'aggiunta di una generale contestualizzazione, e un'analisi del dominio tecnologico, in cui vengono elencate le maggiori tecnologie coinvolte per ogni prodotto secondo la descrizione del capitolato e dalle esperienze pregresse dei componenti del gruppo;

- **Valutazione generale**: composta dagli aspetti positivi e dagli aspetti negativi trovati e discussi dal gruppo riguardo al capitolato in esame;
- **Valutazione finale**: vi si può trovare in breve la motivazione della scelta presa per ogni capitolato in base a ciò che è stato riportato nelle tre sezioni precedenti.

2.1.4 Piano di Progetto

La redazione di un *piano*$_g$ da seguire durante la realizzazione del progetto spetta al *Responsabile*, aiutato nelle scelte dagli *Amministratori*. Il documento dovrà coprire le seguenti tematiche:

- **Analisi dei rischi**: riporta una dettagliata analisi dei rischi che si potrebbero incontrare durante la realizzazione del progetto, determinandone, in base alle conoscenze pregresse e alle nuove acquisite, la probabilità che essi accadano e la loro gravità. Inoltre, quando possibile, verranno analizzati i possibili metodi per affrontarli;
- **Pianificazione**: viene presentata una pianificazione delle *attività*$_g$ da svolgere nel corso del progetto, fornendo delle scadenze temporali il più possibile precise e veritiere;
- **Assegnazione risorse**: riporta la suddivisione oraria delle risorse, tenendo conto dei ruoli ricoperti da tutti i membri. In questo modo si ha una visione chiara su come il gruppo sta procedendo e quante ore vengono impiegate per ogni attività;
- **Preventivo**: sulla base della pianificazione, viene stimata la quantità di lavoro necessaria per portare a termine ogni attività (e quindi ogni fase) del progetto, per arrivare infine ad avere una valutazione complessiva per tutto il progetto e proporre un preventivo finale con il costo del lavoro precedentemente stimato.

2.1.4.1 Strumenti di supporto

2.1.4.1.1 Gantt Project

Per quanto riguarda la creazione dei diagrammi di Gantt abbiamo deciso di utilizzare Gantt Project data la sua completezza, facilità di utilizzo e la possibilità di esportare i diagrammi sotto forma di *PNG*$_g$. Inoltre, i file di Gantt Project sono salvati in formato *XML*$_g$, quindi facilmente versionabili.

2.1.4.1.2 Microsoft Excel

Per la realizzazione dei consuntivi abbiamo deciso di utilizzare Microsoft Excel come strumento per creare i grafici, data la sua efficacia e facilità di utilizzo.

2.1.5 Piano di Qualifica

Il compito di *verifica*$_g$ e *validazione*$_g$ viene svolto da parte dei *Verificatori*. Questa mansione verrà svolta secondo un preciso metodo che deve coprire le seguenti tematiche:

- **Metodo di verifica**: riporta le procedure di controllo sulla *qualità*$_g$ di *processo*$_g$ e di *prodotto*$_g$, considerando i mezzi e le risorse a disposizione;
- **Misure e metriche**: presenta criteri oggettivi per i documenti, i processi e il software;
- **Gestione della revisione**: precisa nel dettaglio le metodologie di comunicazione delle procedure di controllo per la qualità di processo e delle anomalie;
- **Pianificazione del collaudo**: definisce dettagliatamente le metodologie di collaudo a cui sarà sottoposto il progetto realizzato;
- **Resoconto dell'attività di verifica**: riporta le metriche calcolate e il resoconto sul collaudo delle attività sottoposte a verifica e validazione.

2.1.6 Ingresso alle revisioni

Prima di ogni revisione è fondamentale allestire ed organizzare correttamente il materiale da consegnare. Compito del *Responsabile di Progetto* è redigere la lettera di presentazione da includere anch'essa per l'ingresso alla revisione.

2.2 Sviluppo

Il processo in questione affronta le attività ed i compiti svolti dal gruppo con l'obiettivo di sviluppare il software richiesto dal proponente. Per una corretta implementazione è fondamentale:

- Realizzare un prodotto finale conforme alle richieste del proponente;
- Realizzare un prodotto finale che soddisfa i test di verifica e validazione;
- Fissare gli obiettivi di sviluppo;
- Fissare i vincoli tecnologici e di design.

Inoltre, il gruppo ha deciso di seguire le linee guida dettate dallo standard *ISO*$_g$/*IEC*$_g$ 12207. Per questo motivo le attività scelte alla base del progetto di sviluppo saranno le seguenti:

- Analisi dei Requisiti;
- Progettazione;
- Codifica.

2.2.1 Analisi dei requisiti

2.2.1.1 Scopo

Determinare con precisione i requisiti del progetto ed elencarli in modo formale. Essi vengono estrapolati da varie fonti:

- Documenti di specifica del *capitolato*$_g$;
- Incontri con l'azienda proponente;

- Verbali interni ed esterni;
- Casi d'uso.

2.2.1.2 Casi d'uso

Ogni caso d'uso è descritto dalla seguente struttura:

- Codice identificativo:

  UC {codice_padre}.{codice_figlio}

  - UC specifica che si tratta di un caso d'uso;
  - Codice_padre identifica univocamente i casi d'uso;
  - Codice_figlio è un numero progressivo che identifica i sottocasi.
- Titolo;
- Diagramma UML;
- Attori;
- Attori secondari (se presenti);
- Scopo e descrizione;
- Precondizione;
- Scenario principale;
- Postcondizioni;
- Inclusioni (se presenti);
- Estensioni (se presenti).

2.2.1.3 Requisiti

Ogni requisito è descritto dalla seguente struttura:

- Nome;
- Tipo;
- Importanza;
- Stato implementazione;
- Fonti.

Inoltre, a ciascun requisito corrisponde un codice identificativo cosi composto:

R {importanza}.{tipo}.{identificativo}

- R specifica che si tratta di un requisito ;

- Importanza identifica la rilevanza del requisito e può assumere 3 valori:
  - 0: indica che il requisito è obbligatorio e il suo soddisfacimento dovrà necessariamente avvenire;
  - 1: indica che il requisito è desiderabile, cioè il suo soddisfacimento può portare maggiore completezza al sistema ma non è fondamentale per lo stesso;
  - 2: indica che il requisito è opzionale, e quindi la decisione di implementarlo o meno verrà presa dopo le dovute considerazioni;
- Tipo distingue se si tratta di un requisito funzionale (F), di qualità (Q), di prestazione (P) o di vincolo (V);
- Identificativo è un numero progressivo che identifica i sottocasi.

2.2.1.4 UML

I diagrammi UML devono essere realizzati usando la versione del linguaggio *v2.0*

2.2.1.5 Strumenti di supporto

2.2.1.5.1 Trender

Per quanto riguarda il tracciamento dei requisiti, il gruppo ha deciso di utilizzare inizialmente uno strumento non totalmente adatto a questo compito, cioè un foglio di calcolo. Essendo a conoscenza delle limitate funzionalità offerte da questa tecnologia per lo scopo a noi d'interesse, il team 7DOS ha deciso di migrare i dati raccolti in uno strumento creato appositamente per questo scopo. Lo strumento scelto è Trender e rende il tracciamento dei requisiti semi-automatico, inoltre offre la creazione di file .tex per il tracciamento dei requisiti.

2.2.1.5.2 Astah UML

Per quanto concerne la creazione e la modellazione dei diagrammi UML per i casi d'uso, abbiamo deciso di utilizzare Astah UML, data la sua facilità di utilizzo e compatibilità con UML 2.0. Per la successiva fase di modellazione delle classi, potrebbe verificarsi una scelta diversa riguardo al software da utilizzare.

2.2.2 Progettazione

La presente sezione sarà soggetta ad incrementi durante la fase di "Progettazione di dettaglio e codifica". Per questo motivo, non si pone l'obiettivo di risultare completa in questa fase del progetto.

2.2.2.1 Scopo

La Progettazione esplicita concretamente una prima forma ad alto livello del design del software pensata per il progetto, determinandone le caratteristiche più in evidenza, in modo

da comunicare con gli stakeholder e dare delle informazioni chiare, così da intraprendere le prime discussioni sul progetto. I *Progettisti* hanno il compito di definire il sistema in modo esplicito così da poter fare importanti considerazioni riguardo a performance, affidabilità e manutenibilità, capire come è organizzato il sistema e come i componenti interoperano tra di loro sottolineando la possibilità di riutilizzo in larga scala di parte del codice, dato che abbiamo notato una frequente ripetizione dei requisiti in un sistema complesso come quello affrontato dal nostro team. L'architettura dovrà avere le seguenti caratteristiche:

- **Sufficienza**: deve essere in grado di soddisfare i requisiti decritti nel documento *Analisi dei Requisiti*;
- **Comprensibilità**: deve essere capita dagli stakeholder;
- **Flessibilità**: deve permettere modifiche in seguito a delle variazioni, senza essere profondamente ristrutturata;
- **Modularità**: deve essere composta di parti chiare e ben definite;
- **Riusabilità**: le parti devono poter essere usate in più applicazioni;
- **Efficienza**: deve ridurre al minimo gli sprechi;
- **Affidabilità**: deve rispettare le specifiche nel tempo;
- **Disponibilità**: non deve essere inabilitata durante la manutenzione;
- **Sicurezza**: non deve essere vulnerabile ad intrusioni e malfunzionamenti;
- **Semplicità**: deve contenere solo il necessario;
- **Incapsulazione**: le informazioni interne delle componenti devono essere nascoste;
- **Coesione**: le parti che condividono lo stesso fine devono stare insieme;
- **Basso accoppiamento**: parti diverse devono essere poco dipendenti dalle altre.

2.2.2.2 Attività

Il design del software non è composto da una singola attività, bensì da un insieme preciso di sotto-attività progettuali che portano al design finale. Le sotto-attività che il nostro gruppo ha scelto di svolgere sono:

- **Progettazione architetturale**: i progettisti identificheranno una struttura globale del sistema, quindi i componenti principali, le loro relazioni e come saranno distribuiti;
- **Progettazione dell'interfaccia**: i progettisti definiranno le interfacce tra i *moduli*$_g$ di sistema. Ciò deve essere fatto assolutamente in modo non ambiguo dato che, grazie a queste specifiche, ogni componente potrà usare in modo appropriato le funzionalità degli altri componenti del progetto senza sapere l'effettiva implementazione;
- **Selezione e progettazione dei componenti**: i progettisti cercheranno dei componenti riutilizzabili e valuteranno se inserirli nel progetto, specificandone gli appropriati utilizzi e aggiustamenti, o se sarà più opportuno costruire dei nuovi componenti.

2.2.2.3 Diagrammi

Per rendere più chiare le scelte progettuali adottate e ridurre le possibili ambiguità, verranno impiegate le seguenti tipologie di rappresentazioni:

- **Diagrammi delle classi**: per descrivere gli oggetti presenti nel sistema;
- **Diagrammi dei package**: per descrivere le dipendenze delle classi raggruppate in package;
- **Diagrammi di sequenza**: per descrivere la collaborazione nel tempo tra un gruppo di oggetti;
- **Diagrammi di attività**: per descrivere la logica procedurale.

2.2.2.4 Diagrammi delle classi

Lo scopo dei diagrammi delle classi è:

- Fornire la progettazione della visione statica di un'applicazione;
- Descrivere le responsabilità di un sistema;
- Base per diagrammi dei componenti e di rilascio;
- *Reverse Engineering*$_g$.

I diagrammi delle classi sono collegati fra loro da frecce che ne esplicitano le dipendenze. Verranno utilizzati i seguenti tipi di freccia:

- Indica che la classe A ha fra i propri campi dati una o più istanze della classe B;

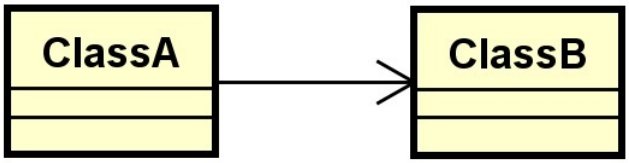

Figura 1: Relazione di dipendenza forte tra classi

- Indica che A dipende da B secondo una primitiva;

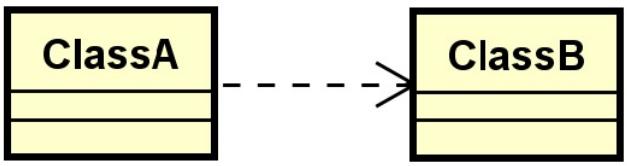

Figura 2: Relazione di dipendenza debole tra classi

- Indica un'aggregazione da A verso B, cioè una relazione non forte nella quale le classi parte hanno un significato anche senza che sia presente la classe tutto;

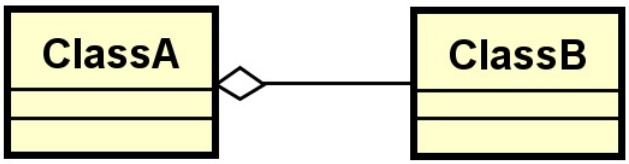

Figura 3: Relazione di aggregazione tra classi

- Indica una composizione da A verso B, cioè una relazione forte nella quale le classi parte hanno un reale significato solo se sono legate alla classe tutto;

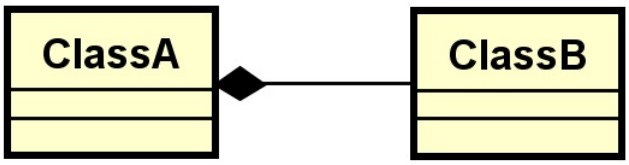

Figura 4: Relazione di composizione tra classi

- Indica l'ereditarietà, cioè che ogni oggetto di A è anche un oggetto di B.

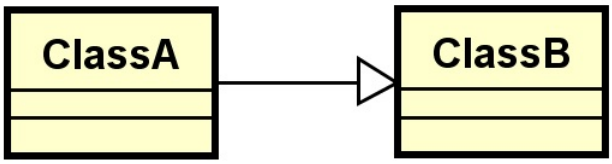

Figura 5: Relazione di ereditarietà tra classi

2.2.2.5 Diagrammi dei package

I package sono rappresentati in forma di rettangoli e dispongono di un nome univoco. Ogni rettangolo contiene i diagrammi delle classi presenti all'interno di quel package. Le dipendenze sono indicate da frecce tratteggiate. Nel seguente esempio si può vedere che il package A ha una dipendenza nei confronti di B.

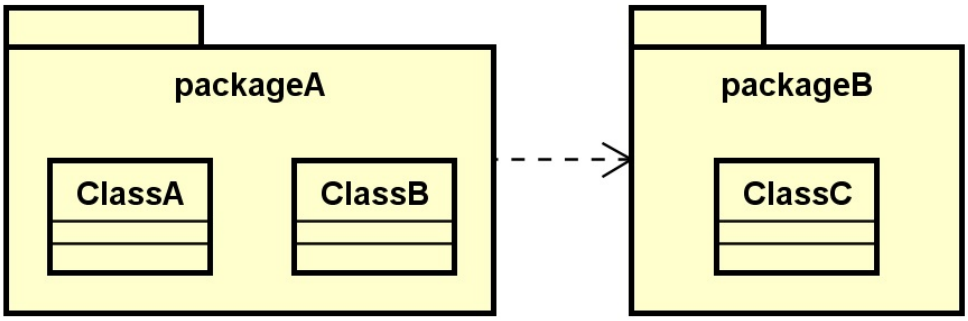

Figura 6: Relazione di dipendenza tra package

2.2.2.6 Diagrammi di sequenza

Questi diagrammi hanno un verso di lettura dall'alto verso il basso, che indica lo scorrere del tempo. Gli oggetti presenti nei diagrammi di sequenza sono rappresentati da rettangoli, al cui interno è presente un nome per identificarli. Sotto ad ogni istanza è presente una linea tratteggiata verticale, che indica la vita dell'oggetto. Da esse possono partire o arrivare delle frecce orizzontali di attivazione, che si distinguono nelle seguenti tipologie:

- Messaggi sincroni, in cui il chiamante resta in attesa di una risposta

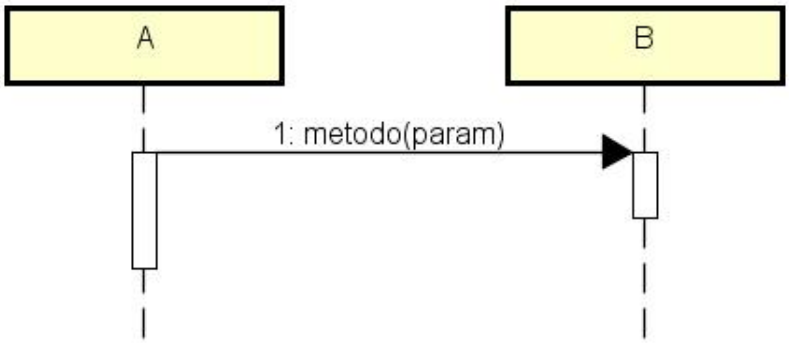

Figura 7: Messaggio sincrono

- Messaggi asincroni, in cui il chiamante non rimane in attesa di una risposta

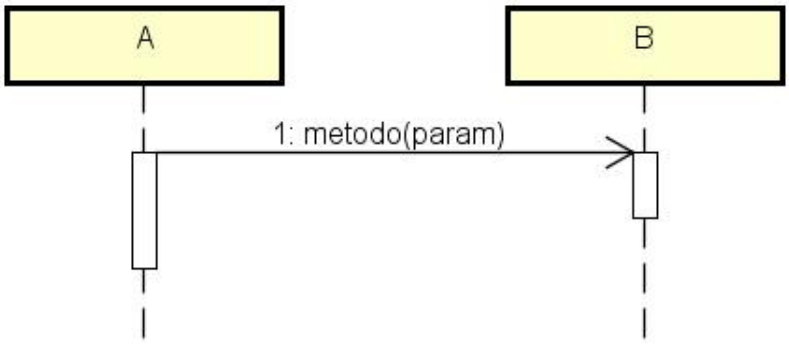

Figura 8: Messaggio asincrono

- Messaggio di ritorno

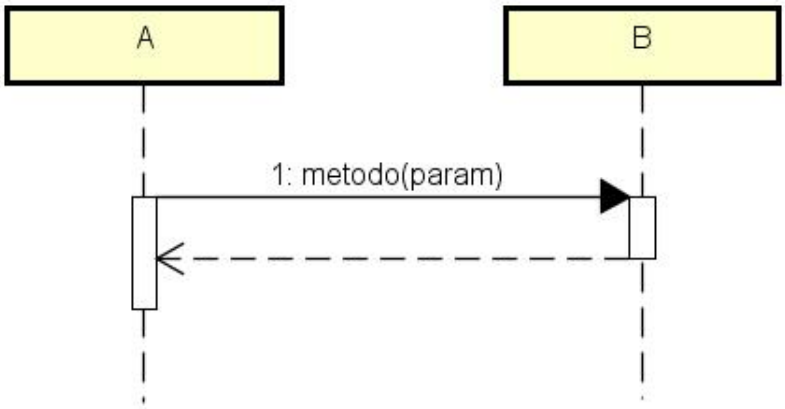

Figura 9: Messaggio di ritorno

- Creazione e distruzione di un partecipante

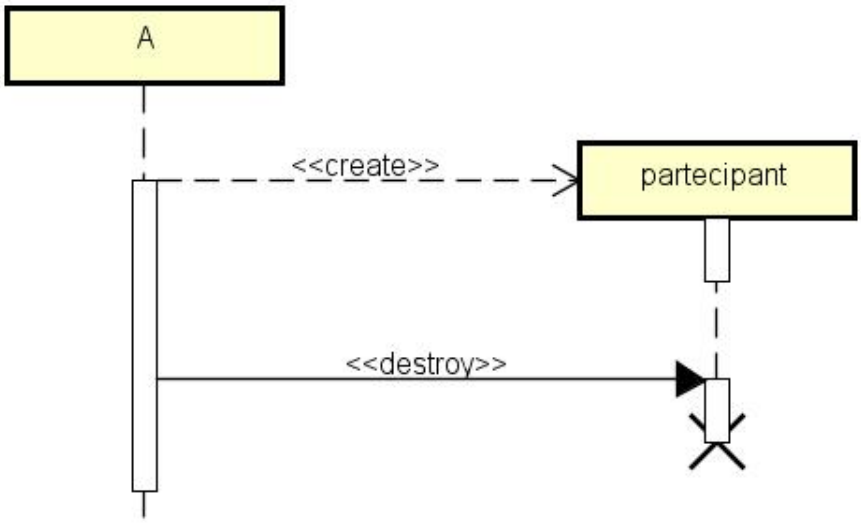

Figura 10: Messaggio di ritorno

2.2.2.7 Diagrammi di attività

I diagrammi di attività aiutano a descrivere gli aspetti dinamici dei casi d'uso e supportano l'elaborazione parallela. Ogni attività viene rappresentata da un rettangolo dagli angoli arrotondati e contiene al suo interno il nome. Un diagramma si compone delle seguenti parti:

- Nodo iniziale da dove inizia l'esecuzione del processo

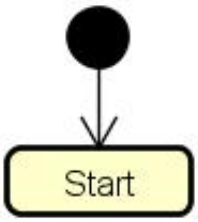

Figura 11: Nodo iniziale

- Fork per l'elaborazione parallela

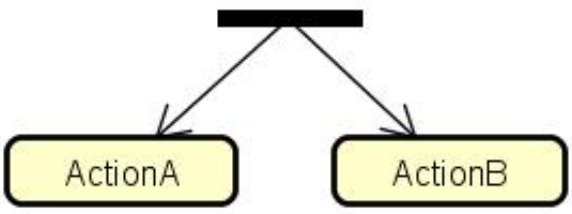

Figura 12: Fork di attività

- Join per la sincronizzazione dei processi paralleli

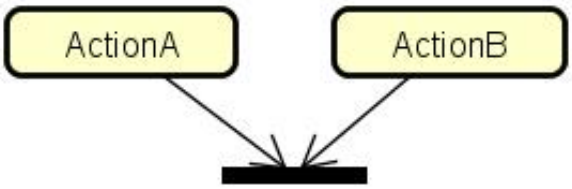

Figura 13: Join dei processi paralleli

- Branch per la scelta di uno dei rami decisionali

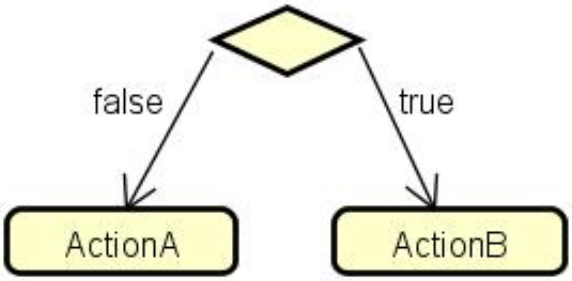

Figura 14: Branch per decisione

- Merge per unire i rami separati dal branch

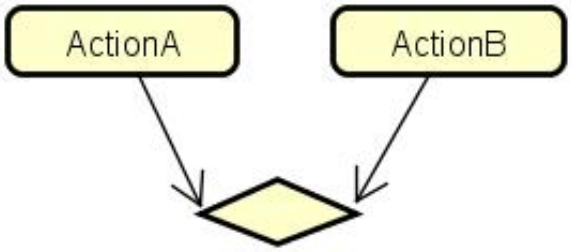

Figura 15: Merge per unire rami separati da branch

- I segnali seguono la notazione seguente per indicare il tempo da aspettare prima dell'invio di un segnale, l'invio di un segnale e la sua ricezione

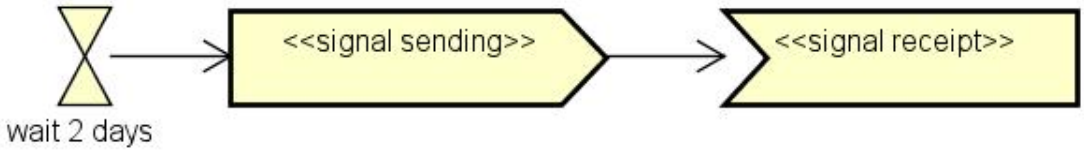

Figura 16: Notazioni dei segnali

- Nodo di fine flusso per indicare la terminazione di un ramo

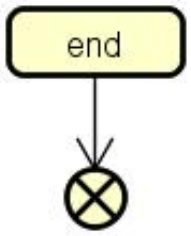

Figura 17: Nodo di fine flusso

- Nodo finale per indicare la terminazione dell'esecuzione

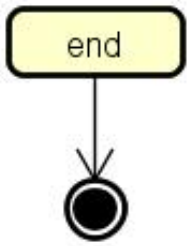

Figura 18: Nodo finale

2.2.2.8 Tecnologie

Lo sviluppo del progetto richiede l'utilizzo di tecnologie, alcune imposte del capitolato, altre soggette a libera scelta da parte del team. Le tecnologie il cui uso è obbligatorio sono:

- **Grafana;**
- **AngularJS;**
- **InfluxDB.**

Le tecnologie che invece il team ha scelto a valle di un attività di comparazione sono:

- **TypeScript**: per la scrittura del plug-in per Grafana;
- **Webpack**: per il processo di build del plug-in;
- **JsBayes**: per il calcolo delle probabilità delle reti Bayesiane;
- **NodeJS**: per l'esecuzione lato server di codice JavaScript.

2.2.2.9 Integrazione continua

Il team ha scelto di applicare pratiche di integrazione continua e per farlo ha scelto di usare

TravisCI$_g$. TravisCI è una piattaforma che permette di associare un repository GitHub ad un processo di build e ad un set di test di unità. Ad ogni commit viene eseguito il processo di build e vengono eseguiti i test. TravisCI inoltre offre integrazione con *Coveralls*$_g$ un servizio per la gestione del test coverage.

2.2.3 Codifica

La seguente sotto-sezione ha lo scopo di elencare le norme di codifica alle quali i Programmatori devono aderire durante l'attività di programmazione. Gli argomenti trattati riguardano:

- Norme generali dello stile di codifica da adottare, qualsiasi sia il linguaggio utilizzato;
- Norme specifiche per i linguaggi *JavaScript*$_g$ *TypeScript*$_g$ *HTML*$_g$ e *CSS*$_g$.

Il gruppo 7DOS ha deciso di porre delle norme sulla codifica in quanto vantaggiose per la generazione di codice leggibile, adatto per le fasi di verifica e manutenzione, e per garantire qualità al prodotto. Soltanto il *Responsabile di Progetto*, dopo un'attenta analisi e valutazione, potrà ammettere modifiche alle convenzioni stabilite.

2.2.3.1 Norme sul codice

- **Nomi**: i nomi di variabili, classi, funzioni e metodi dovranno essere in *camel case*$_g$ ed in lingua inglese. I nomi di variabili, metodi e funzioni dovranno avere la prima lettera minuscola mentre i nomi delle classi dovranno avere la prima lettera maiuscola.
  Ogni nome deve essere unico e illustrativo, cioè deve richiamare lo scopo per cui è stato dichiarato.
- **Ricorsione**: la ricorsione va evitata quando possibile. Per ogni funzione ricorsiva sarà necessario fornire una prova di terminazione e sarà necessario valutare il costo in termini di occupazione della memoria. Nel caso l'utilizzo di memoria risulti troppo elevato la ricorsione verrà rimossa;
- **Intestazioni**

  Ogni file contenente codice deve essere provvisto di un'intestazione contenente:

```
/*
@File:  nome del file
@Version:  versione del file
@Type:  tipo del file
@Author:  autore
@Email:  indirizzo email dell'autore
@Date:  data di creazione
@Desc:  descrizione del file

@Changelog:  autore, data, descrizione
*/
```

Ogni classe deve essere provvista di un'intestazione contenente:

```
/*
@Class:  nome della classe
@Desc:  descrizione della classe
*/
```

Ogni funzione e metodo devono essere provvisti di un'intestazione contenente:

```
/*
@Desc:  descrizione della funzione/metodo
@Param:  elenco dei parametri
@Return:  valore ritornato
*/
```

- **Commenti**: i commenti devono essere esaustivi e non prolissi in modo da aiutare il lettore a comprendere al meglio ciò che si sta leggendo. I commenti devono essere scritti in inglese. Inoltre è consentito l'uso di particolari tipi di commenti che hanno l'obiettivo di favorire la stesura del codice come:
  - TODO: per segnalare la presenza di funzionalità che devono ancora essere implementate;
  - FIX: per segnalare la presenza di un errori che devono ancora essere corretti.
- **Versionamento:** la versione del file, inserita all'interno dell'intestazione, segue la convenzione **X.Y**.
  - **X**: indica l'indice di versione principale. Incrementare il suo valore indica che sono state apportate modifiche significative e stabili al file, comportando di conseguenza l'azzeramento dell'indice Y;
  - **Y**: indica l'indice di versione parziale. Incrementare il suo valore indica che sono state apportate modifiche minori, come l'aggiunta o la rimozione di un metodo.

  Entrambe gli indici partono da 0. La versione *1.0* indica la prima versione stabile del file, la quale copre le funzionalità obbligatorie.

2.2.3.2 JavaScript

Le convenzioni stilistiche definite in *JavaScript Style Guide and Coding Conventions*$_g$[1] verranno seguite dai *Programmatori* per lo sviluppo dell'intero progetto.

- **Indentazione**: ogni livello di indentazione prevede l'uso di due spazi;

```
1    function toString() {
2    ..let example;  // CORRECT
3    }
```

[1] https://www.w3schools.com/js/js_conventions.asp

```
4
5       function toString() {
6       ....let example;  // INCORRECT
7       }
```

- **Spazi**
  - Viene posto uno spazio tra gli operatori (= + - * / < >) e dopo le virgole;

```
1       // CORRECT
2       var example = "7DOS";
3
4       //INCORRECT
5       var example="7DOS";
6
7       // CORRECT
8       var example = ["one", "two", "three"];
9
10      // INCORRECT
11      var example = ["one","two","three"];
```

  - Non vengono posti spazi prima della lista degli argomenti nelle chiamate di funzioni e nelle dichiarazioni.

```
1       // CORRECT
2       function someOperation(int value) {
3       ..var result;
4       }
5
6       // INCORRECT
7       function someOperation (int value) {
8       ..var result;
9       }
```

- **Dichiarazioni di variabili**: ogni dichiarazione di variabile prevede l'utilizzo di *var* o *let* e termina con un punto e virgola;

```
1   // CORRECT
2   var example = "7DOS";
3
4   //INCORRECT
5   var example= "7DOS"
6
7   // CORRECT
8   let example = ["one", "two", "three"];
9
10  // INCORRECT
11  let example = ["one","two","three"]
```

- **Dichiarazione di funzioni**

– La parentesi graffa di apertura viene posta nella stessa riga della dichiarazione della funzione, utilizzando uno spazio prima;

```
1       // CORRECT
2       function toString() {
3       ..var example;
4       }
5
6       // INCORRECT
7       function toString(){
8       ..var example;
9       }
```

– La parentesi graffa di chiusura viene posta in una nuova riga senza utilizzare spazi.

```
1       // CORRECT
2       function toString() {
3       ..var example;
4       }
5
6       // INCORRECT
7       function toString() {
8       ..var example;}
```

- **Cicli**: ogni parametro è separato dal successivo con uno spazio, così come le parentesi tonde. Valgono le stesse regole per le parentesi graffe viste nella dichiarazione di funzioni;

```
1   // CORRECT
2   while (i < 10) {
3
4   }
5
6   //INCORRECT
7   while(i<10)
8   {
9
10  }
11
12  // CORRECT
13  for(i = 0; i < 10; i++) {
14
15  }
16
17  // INCORRECT
18  for(i=0;i<10;i++)
19  {
20
21  }
```

- **Controlli condizionali**: valgono le stesse regole per le parentesi graffe viste nella dichiarazione di funzioni. Ogni *else* successivo ad un *if* deve posizionarsi nella stessa riga della parentesi graffa di chiusura dell'*if*;

```
1  // CORRECT
2  if (i < 10) {
3
4  } else {
5
6  }
7
8  // INCORRECT
9  if (i < 10) {
10
11 }
12 else {
13
14 }
```

- **Lunghezza di una linea**: ogni linea di codice non deve contenere più di 80 caratteri. Se necessario, spezzare la linea dopo punti specifici, come un operatore o una virgola;
- **Lunghezza metodi e funzioni**: il corpo dei metodi e delle funzioni non deve superare le 40 righe di lunghezza e tre gradi indentazione. In questo modo si evita la creazione di strutture troppo complesse, difficili da testare e mantenere. Nel caso un metodo o una funzione dovesse infrangere questi limiti, è opportuno riformattare il corpo utilizzando funzioni ausiliarie. Se ciò non è possibile, il *Programmatore*, con l'aiuto degli altri componenti, cercherà di circoscrivere la complessità del metodo/funzione per evitare che si discosti troppo dai limiti imposti.

2.2.3.3 TypeScript

Le convenzioni stilistiche definite in *Grafana Plugin Code Styleguide*$_g$[2] e, più in particolare per il linguaggio Typescript, in *Angular TypeScript Styleguide*$_g$[3] verranno seguite dai programmatori per lo sviluppo dell'intero progetto.

- **Import**
  - Viene lasciata una linea vuota tra i vari *import* di terze parti;

```
1        // CORRECT
2        import { Example } from "../code";
3
4        import { Library1, Library2, Library3 } from "../code";
5
6        // INCORRECT
7        import{ Component, NgModule, OnInit } from "../code";
```

[2] http://docs.grafana.org/plugins/developing/code-styleguide
[3] https://angular.io/guide/styleguide

```
8         import{ Component, NgModule, OnInit } from "../code";
```

– I moduli presenti nello stesso *import* devono essere listati in ordine alfabetico e staccati con uno spazio sia dalle parentesi graffe e che l'uno dall'altro;

```
1         // CORRECT
2         import { Component, NgModule, OnInit } from "../code";
3
4         //INCORRECT
5         import{Component,NgModule,OnInit} from "../code";
```

– Viene posto uno spazio tra le parentesi graffe che contengono i moduli e la dichiarazione *import*.

```
1         // CORRECT
2         import { Component, NgModule, OnInit } from "../code";
3
4         //INCORRECT
5         import{ Component, NgModule, OnInit } from "../code";
```

- **Export**

– La parentesi graffa di apertura viene posta nella stessa riga della dichiarazione della funzione, utilizzando uno spazio prima;

```
1     //  CORRECT
2     export class JsImport {
3
4     }
5
6     //  INCORRECT
7     export class JsImport
8     {
9
10    }
```

– La parentesi graffa di chiusura viene posta in una nuova riga senza utilizzare spazi.

```
1     //  CORRECT
2     export class JsImport {
3       textJson: string;
4       time: string;
5     }
6
7     //  INCORRECT
8     export class JsImport {
9       textJson: string;
10      time: string;}
```

- **Test**
  - Ogni test deve rispettare la seguente struttura:

```
describe("NomeClasse - NomeMetodo", () => {
  it("Input, Eventuali altre operazioni - Output atteso", () => {
    // primo test del metodo NomeMetodo
  });
  it("Input, Eventuali altre operazioni - Output atteso", () => {
    // secondo test del metodo NomeMetodo
  });
});
```

2.2.3.4 HTML

Le convenzioni stilistiche definite in *HTML5 Style Guide and Coding Conventions*$_g$[4] verranno seguite dai *Programmatori* per lo sviluppo dell'intero progetto.

- **Indentazione**: ogni livello di indentazione prevede l'uso di due spazi;

```
// CORRECT
<body>
..<h1>Heading</ h1>
</ body>

// INCORRECT
<body>
....<h1>Heading</ h1>
</ body>
```

- **Doctype**: la prima riga di ogni documento HTML deve presentare la dichiarazione del tipo di documento;

```
<!doctype html>
```

- **Elementi**
  - **Nomi**: ogni nome deve essere scritto in minuscolo;

```
// CORRECT
<section></ section>

// INCORRECT
<SECTION></ SECTION>
```

[4] https://www.w3schools.com/html/html5_syntax.asp

  - **Utilizzo dei tag**: ogni tag deve sempre essere chiuso. Per compatibilità con i vecchi browser, i tag non vuoti, anche se privi di contenuto, vanno nella forma , mentre quelli vuoti vanno nella forma
.

```
1    // CORRECT
2    <p></ p>
3
4    // INCORRECT
5    <p />
6
7    // CORRECT
8    <br />
9
10   // INCORRECT
11   <br></ br>
```

- **Spazi**: '=' tra l'attributo e il suo valore deve essere racchiuso tra spazi;

```
1  // CORRECT
2  <p class = "...">
3
4  </ p>
5
6  // INCORRECT
7  <p class="...">
8
9  </ p>
```

- **Lunghezza di una linea**: ogni linea di codice non deve contenere più di 80 caratteri;
- **Tag <html>, <head>, <body>**: i tag <html>, <head>, <body> non devono essere mai omessi.

2.2.3.5 CSS

Le convenzioni stilistiche definite in *Airbnb CSS / Sass Styleguide*$_g$[5] verranno seguite dai programmatori per lo sviluppo dell'intero progetto.

- **Formattazione**
  - Le proprietà di stile relative ad un selettore devono essere indentate con due spazi;

```
1      // CORRECT
2      .someselector {
3      ..color: "...";
4      ..text-aling: "...";
5      }
6
```

[5] https://github.com/airbnb/css

```
7       // INCORRECT
8       .someselector {
9       ....color: "...";
10      ....text-aling: "...";
11      }
```

– In caso di selettori multipli per una o più regole, inserire ogni selettore in una linea singola;

```
1     // CORRECT
2     .selectorOne,
3     .selectorTwo,
4     .selectorThree {
5     ..color: "...";
6     ..text-aling: "...";
7     }
8
9     // INCORRECT
10    .selectorOne, .selectorTwo, .selectorThree {
11    ....color: "...";
12    ....text-aling: "...";
13    }
```

- **Spazi**
  – Deve essere posto uno spazio tra un selettore e la parentesi graffa di apertura di una *rule declaration*$_g$;

```
1     .someselector {  // CORRECT
2
3     }
4
5     .someselector  // INCORRECT
6     {
7
8     }
```

  – Nella dichiarazione di una proprietà deve essere posto uno spazio dopo ':' ma non prima;

```
1     .someselector {  // CORRECT
2     ..property: "...";
3     }
4
5     .someselector  // INCORRECT
6     {
7     ..property : "...";
8     }
```

- **Rule declaration**

– La parentesi graffa di chiusura deve essere posta in una nuova linea, senza l'uso di spazi;

```
    .someselector {  // CORRECT
    ..property: "...";
    }

    .someselector  // INCORRECT
    {
    ..property : "...";}
```

– In presenza di più rule declaration, deve essere posta una linea vuota tra esse.

```
    // CORRECT
    .selector1 {
    ..property: "...";
    }

    .selector2 {
    ..property : "...";
    }

    // INCORRECT
    .selector1 {
    ..property: "...";
    }
    .selector2 {
    ..property : "...";
    }
```

– Tutte le proprietà relative ad un selettore devono essere dichiarate in ordine alfabetico.

```
    // CORRECT
    .someSelector {
    ..background: "...";
    ..border: "...";
    ..font-size: "...";
    ..text-align: "...";
    }

    // INCORRECT
    .someSelector1 {
    ..border: "...";
    ..text-align: "...";
    ..background: "...";
    ..font-size: "...";
    }
```

2.2.3.6 Metriche per i prodotti software

Di seguito viene riportato un insieme di metriche relative ai prodotti software ritenute come fondamentali dal team 7DOS. Questa sezione potrebbe essere ampliata successivamente qualora fosse necessario introdurre nuove metriche per la valutazione della qualità dei prodotti software.

2.2.3.6.1 Functional Implementation Completeness

Misurazione in percentuale del grado con cui le funzionalità offerte dalla corrente implementazione del software coprono l'insieme di funzioni specificate nei requisiti.
Abbiamo scelto questa metrica per valutare il grado di completezza del prodotto; l'obiettivo è implementare tutte le funzionalità richieste.
Viene utilizzata la seguente formula:

$$FI_{Comp} = \frac{NF_i}{NF_r} * 100$$

Dove:

- **FI_{Comp}**: è il valore della metrica;
- **NF_i**: è il numero di funzioni attualmente implementate;
- **NF_r**: è il numero di funzioni specificate dai requisiti.

2.2.3.6.2 Average Functional Implementation Correctness

Misurazione in percentuale del grado in cui le funzionalità offerte dalla corrente implementazione del software, in media, rispettano il livello di precisione indicato nei requisiti.
Abbiamo scelto questa metrica per valutare il grado di accuratezza e garantire la qualità dei risultati restituiti dal prodotto, in quanto andrà a fare previsioni sulla verosimiglianza di alcuni eventi in base ai dati forniti, ed è necessario che tali previsioni siano sufficientemente accurate.
Viene utilizzata la seguente formula:

$$aFI_{Corr} = \frac{\sum_{i=1}^{N} \frac{iPF_i}{rPF_i}}{N} * 100$$

Dove:

- **aFI_{Corr}**: è il valore della metrica;
- **iPF_i**: è il livello di precisione della i-esima funzione implementata;
- **rPF_i**: è il livello di precisione della i-esima funzione secondo i requisiti;
- **N**: è il numero totale di funzioni considerate.

2.2.3.6.3 Tempo di risposta

Misurazione in secondi che indica il tempo medio che intercorre fra la richiesta software di una determinata funzionalità e la restituzione del risultato all'utente.
Viene utilizzata la seguente formula:

$$TR = \frac{\sum_{i=1}^{N} T_i}{N}$$

Dove:

- **TR**: è il valore della metrica;
- $\mathbf{T_i}$: è il tempo intercorso fra la richiesta i di una funzionalità ed il comportamento delle operazioni necessarie a restituire un risultato a tale richiesta;
- **N**: è il numero totale di funzioni considerate.

2.2.3.6.4 Average Learning Time

Misurazione in minuti del tempo medio impiegato da un utente per imparare ad utilizzare una singola funzionalità del prodotto.
Abbiamo scelto questa metrica poiché, trattandosi di un prodotto che verrà reso disponibile pubblicamente, abbiamo ritenuto importante renderlo semplice da imparare per permetterne l'uso ad una vasta gamma di utenti.
Viene utilizzata la seguente formula:

$$aLT = \frac{\sum_{i=1}^{N} LT_i}{N}$$

Dove:

- **aLT**: è il valore della metrica;
- $\mathbf{LT_i}$: è il tempo necessario ad imparare ad utilizzare la i-esima funzione implementata, espresso in minuti;
- **N**: è il numero totale di funzioni considerate.

2.2.3.6.5 Failure Density

Misurazione in percentuale della quantità di failure rilevati rispetto alla quantità di test eseguiti.
Abbiamo scelto questa metrica per garantire che il prodotto sia generalmente stabile e non risulti poco utilizzabile o inutilizzabile a causa di eccessive $failure_g$. Valutazioni più precise saranno effettuate in base ai singoli risultati dei test.
Viene utilizzata la seguente formula:

$$FD = \frac{T_f}{T_e} * 100$$

Dove:

- **FD**: è il valore della metrica;
- $\mathbf{T_f}$: è il numero di test falliti;
- $\mathbf{T_e}$: è il numero di test eseguiti.

2.2.3.6.6 Operazioni con gestione errori

Misurazione in percentuale della quantità di funzionalità in grado di gestire in modo corretto possibili errori.
Viene utilizzata la seguente formula:

$$BONC = \frac{NF_E}{NO_T} * 100$$

Dove:

- **BONC**: è il valore della metrica;
- $\mathbf{NF_E}$: è il numero di failure evitate;
- $\mathbf{NO_T}$: è il numero di test con possibile esecuzione di operazioni non corrette.

2.2.3.6.7 Failure Analysis

Misurazione in percentuale delle modifiche effettuate in risposta a failure che hanno portato all'introduzione di nuove failure in altre componenti del sistema.
Viene utilizzata la seguente formula:

$$FA = \frac{NF_I}{NF_R} * 100$$

Dove:

- **FA**: è il valore della metrica;
- $\mathbf{NF_I}$: è il numero di failure delle quali sono state individuate le cause;
- $\mathbf{NF_R}$: è il numero di failure rilevate.

2.2.3.6.8 Comment Ratio

Misurazione in percentuale che indica il rapporto tra il numero di righe in cui è presente del codice ed il numero di righe in cui sono presenti dei commenti(o annotazioni).
Viene utilizzata la seguente formula:

$$CR = \frac{NR_C}{NR_A} * 100$$

Dove:

- **CR**: è il valore della metrica;
- $\mathbf{NR_C}$: è il numero di righe in cui è presente del codice;
- $\mathbf{NR_A}$: è il numero di righe in cui sono presenti dei commenti.

3 Processi di supporto

3.1 Documentazione

Questa sezione descrive in modo dettagliato le procedure adottate dal gruppo in merito alla redazione, verifica e approvazione di tutta la documentazione prodotta. Tali norme devono essere rispettate in modo tassativo, al fine di realizzare documenti formali, coerenti e non ambigui.

3.1.1 Fasi di sviluppo

Ogni documento deve attraversare le seguenti fasi per essere considerato formale:

- **Redazione**: un documento si trova in questa fase dal momento in cui viene creato fino alla sua approvazione. I *Redattori* si occupano della stesura e della modifica delle sezioni che gli sono state assegnate dal *Responsabile di Progetto*;
- **Verifica**: un documento entra in questa fase al termine del lavoro dei *Redattori*. Il *Responsabile di Progetto* deve assegnare ai *Verificatori* la procedura di verifica e validazione del documento. In caso di esito positivo esso passa allo stato *Approvato*, nel caso contrario il *Responsabile di Progetto* affida ai *Redattori* il compito di apportare eventuali correzioni;
- **Approvazione**: un documento entra in questa fase una volta superata la fase di verifica in modo positivo ed è compito del *Responsabile di Progetto* approvarlo in maniera ufficiale.

3.1.2 Template

Per uniformare la struttura dei documenti abbiamo creato un template $LaTeX_g$ che implementa la formattazione e l'impaginazione degli stessi. Il contenuto di ogni documento è costituito di più file, uno per ogni sezione, la cui stesura è stata incaricata ai *Redattori*.

3.1.3 Struttura dei documenti

Tutta la documentazione deve rispettare la medesima struttura.

3.1.3.1 Frontespizio

Questa sezione contiene tutti gli elementi che dovranno essere presenti nella prima pagina di ogni documento.

- **Logo del gruppo**;
- **Titolo del documento**;
- **Nome del gruppo**;
- **Data di approvazione**;
- **Informazioni sul documento**:

  - Versione corrente;
  - Nome e cognome del *Responsabile di Progetto*;
  - Nome e cognome dei *Verificatori*;
  - Nome e cognome dei *Redattori*;
  - Destinazione d'uso;
  - Destinatari del documento;
  - Indirizzo email del gruppo.
- **Descrizione del documento**.

3.1.3.2 Diario delle modifiche

Questa sezione contiene le modifiche apportate al documento, organizzate in modo tabulare ed ordinate in modo decrescente dall'alto verso il basso secondo la versione dello stesso. Ogni colonna descrive le seguenti informazioni:

- **Versione**: versione del documento;
- **Data**: data di esecuzione;
- **Descrizione**: tipo e soggetto di ogni modifica;
- **Autore**: nome e cognome dell'autore;
- **Ruolo**: ruolo dell'autore in quel momento.

3.1.3.3 Indice

Ogni documento, esclusi i verbali, contiene un indice, il quale consente una visione generale del suo contenuto, ordinato e numerato rispetto alle sezioni presenti. Ci possono essere al più tre tipologie: indice delle sezioni, indice delle tabelle e indice delle immagini.

3.1.3.4 Intestazione

- Logo del gruppo a sinistra;
- Titolo della sezione corrente a destra.

3.1.3.5 Piè di pagina

- Nome del documento e nome del gruppo a sinistra;
- Numero progressivo della pagina corrente a destra.

3.1.4 Norme tipografiche

3.1.4.1 Stile del testo

- **Corsivo**: è usato per termini specifici o poco comuni, per indicare ruoli all'interno del progetto, per le citazioni e per i riferimenti ai documenti. Nel Glossario tutti i termini poco comuni in lingua inglese sono in corsivo;
- **Grassetto**: è usato per evidenziare concetti e parole chiave;
- **Maiuscolo**: è usato per indicare gli acronimi;
- **Sottolineato**: è usato nel Glossario per evidenziare link a termini presenti nel documento che vengono impiegati nella descrizione di altri termini;
- **Azzurro**: è usato per indicare collegamenti ipertestuali;
- **Verde**: è usato per includere frammenti di codice all'interno della documentazione.

3.1.4.2 Elenchi puntati

Ogni punto dell'elenco deve avere la lettera maiuscola e terminare con il carattere punto e virgola, tranne l'ultimo che deve terminare con il carattere punto.

3.1.4.3 Note a piè di pagina

In caso di presenza in una pagina interna di note da esplicare, esse vanno indicate nella pagina corrente, in basso a sinistra. Ogni nota deve riportare un numero e una descrizione.

3.1.4.4 Formati comuni

- **Orari**: **HH:MM**
  - **HH**: indica le ore scritte con due cifre;
  - **MM**: indica i minuti scritti con due cifre.
- **Date**: **AAAA-MM-GG**
  - **AAAA**: indica l'anno scritto con quattro cifre;
  - **MM**: indica il mese scritto con due cifre;
  - **GG**: indica il giorno scritto con due cifre.
- **Termini ricorrenti**
  - **Nomi propri**: vanno scritti usando il formato **Nome Cognome**;
  - **Ruoli di progetto**: vanno scritti in corsivo e con la lettera maiuscola;
  - **Nomi dei documenti**: vanno scritti con la lettera maiuscola;
  - **Riferimenti ai documenti**: vanno scritti in corsivo e con la lettera maiuscola.

3.1.5 Elementi grafici

3.1.5.1 Immagini

Le immagini devono essere separate dal testo lasciando una spaziatura per facilitarne la lettura e vanno centrate orizzontalmente. Possono essere organizzate in modo affiancato e per ogni figura deve essere presente una breve didascalia, oltre ad un identificativo che permetta il relativo inserimento all'interno dell'indice delle immagini. I formati consentiti sono **PNG** e **JPG**.

3.1.5.2 Tabelle

Le tabelle devono essere separate dal testo lasciando una spaziatura per facilitarne la lettura e vanno centrate orizzontalmente. Possono essere organizzate in modo affiancato e per ogni tabella deve essere presente una breve didascalia, oltre ad un identificativo che permetta il relativo inserimento all'interno dell'indice delle tabelle. L'intestazione di ogni colonna deve essere in grassetto ed avere la lettera maiuscola.

3.1.6 Nomenclatura dei documenti

Il formato usato per la nomenclatura dei documenti, tranne i *Verbali*, è il seguente:

NomeDocumento_vX.Y.Z

- **NomeDocumento**: indica il nome del documento, scritto senza spazi e con la lettera maiuscola in ogni parola.
- **vX.Y.Z**: indica la versione del documento secondo il seguente criterio:
  - **X**: viene incrementato in seguito ad un'approvazione ufficiale del documento da parte del *Responsabile di Progetto* e ciò comporta l'azzeramento di Y e Z;
  - **Y**: viene incrementato in seguito ad un'azione di verifica o di stesura di una parte corposa del documento e ciò comporta l'azzeramento di Z;
  - **Z**: viene incrementato in seguito ad un'azione di stesura di una parte esigua del documento.

Il formato usato per la nomenclatura dei *Verbali* invece è **Verbale_AAAA-MM-GG**.

3.1.7 Classificazione dei documenti

3.1.7.1 Documenti informali

Un documento viene considerato informale se non è stato approvato dal *Responsabile di Progetto*, pertanto è concesso esclusivamente ad uso interno al gruppo.

3.1.7.2 Documenti formali

Un documento viene considerato formale dopo aver superato con esito positivo l'attività di

verifica e in seguito alla sua appovazione da parte del *Responsabile di Progetto*, pertanto può essere destinato ad una distribuzione esterna al gruppo.

3.1.7.3 Verbali

Ogni verbale deve essere redatto dal segretario durante le riunioni, sia interne che esterne, tenute dal gruppo e deve rispettare il seguente contenuto:

- **Informazioni incontro**: informazioni generali riguardo la riunione descritte secondo la seguente struttura:
  - Luogo;
  - Data;
  - Ora;
  - Partecipanti del gruppo;
  - Partecipanti esterni.
- **Argomenti affrontati**: descrizione dei temi discussi dal gruppo ed eventuali decisioni prese a riguardo.

3.1.8 Sigle usate

- **AR**: Analisi dei Requisiti;
- **GL**: Glossario;
- **NdP**: Norme di Progetto;
- **PdP**: Piano di Progetto;
- **PdQ**: Piano di Qualifica;
- **SF**: Studio di Fattibilità;
- **RA**: Revisione di Accettazione;
- **RP**: Revisione di Progettazione;
- **RQ**: Revisione di Qualifica;
- **RR**: Revisione dei Requisiti.

3.1.9 Glossario

Il Glossario deve contenere tutti quei termini che possono risultare ambigui o che possono essere fraintesi. Le definizioni, elencate in modo alfabetico, devono essere chiare e concise. L'inserimento nel Glossario deve avvenire parallelamente alla stesura della documentazione. Deve essere segnalata soltanto la prima occorrenza del termine in un documento e deve essere impiegato l'apposito comando personalizzato che lo formatta in corsivo aggiungendo una lettera 'g' come pedice(eg. *Termine nel Glossario*$_g$).

3.1.10 Strumenti di supporto

3.1.10.1 LaTeX

L'intera documentazione deve essere redatta usando il linguaggio di markup LaTeX che offre la possibilità di personalizzare comandi e variabili da usare all'interno dei documenti facilitandone la gestione e l'aggiornamento. Permette anche di dividere il contenuto di un documento dalla rispettiva struttura, suddividendolo in sezioni per facilitarne la stesura da parte dei *Redattori* e di mantenere la stessa formattazione in tutti i documenti mediante l'uso di un template comune.

3.1.10.2 TexStudio

Per redigere i documenti in LaTeX deve essere impiegato l'editor TexStudio. L'ambiente mette a disposizione un'interfaccia per la scrittura dei documenti, organizzati secondo una struttura gerarchica e la visualizzazione dell'anteprima del risultato ottenuto in seguito alla compilazione avvenuta con successo e in caso contrario segnalando all'utente eventuali errori.

3.2 Versionamento

Il servizio di hosting scelto per la repository è Github. La repository è privata e sarà resa pubblica a fine progetto. Seguono le norme che ne riguardano gli aspetti principali.

3.2.1 Norme sui file

I file devono rispettare le norme di nomenclatura già specificate in 3.1.6 Nomenclatura dei documenti. Tramite il file .gitignore vengono esclusi i file intermedi di compilazione dei file latex. Gli unici file ammessi nella repository sono .tex .jpg .png .pdf.

3.2.2 Norme sui *Commit*$_g$

Ogni modifica sostanziale effettuata ai file dev'essere seguita da un commit. Il commento associato ad ogni commit deve essere nella forma: [nome file] descrizione. La descrizione deve essere concisa e quanto più esaustiva possibile. Ad ogni commit deve corrispondere un aggiornamento del diario delle modifiche.

3.2.3 Norme su branching e merging

Le norme usate per branching e merging si rifanno al Git Feauture Branch Workflow: master è il branch in cui risiederà documentazione e codice verificati e approvati. Le modifiche o le nuove feature vanno implementate su un feature branch che saranno verificati prima di effettuare una pull request verso master. La pull request permette di non modificare master fintanto che non viene verificato il branch. Ogni nuovo feature branch deve:

- Essere creato a partire da master;
- Avere un nome che permetta di capire a quale feature è associato.

Queste norme saranno rivisitate all'inizio del processo di sviluppo per accertarsi che il feature branch workflow sia il più opportuno da usare. Una possibile alternativa con cui verrà confrontato è GitFlow.

3.2.4 Strumenti di supporto

3.2.4.1 Github

Github è il servizio di hosting scelto per la repository. Funziona utilizzando git, un software di *versionamento*$_g$ descritto nella sezione immediatamente successiva. Oltre all'hosting del codice sorgente, offre altre caratteristiche utili, tra le quali:

- Un *Issue tracking System*$_g$, con etichette e *milestone*$_g$;
- Una lista dei commit passati e delle relative modifiche ai file semplice da consultare;
- Account Educational gratuito per gli studenti, il quale permette la creazione di repository private, normalmente creabili solo con un account a pagamento.

3.2.4.2 Git

Git è un software open-source di controllo *versione*$_g$ distribuito. Viene usato con Github, in quanto è un servizio già conosciuto dai membri del team e molto comodo da utilizzare.

3.2.4.3 Client Git

A seconda delle preferenze, ogni membro del team deciderà se usare come client GitKraken o Github Desktop. Entrambi sono client desktop che facilitano l'utilizzo di git, rendendo molto più semplice inviare le modifiche locali al repository remoto, o viceversa.

3.2.5 Metriche per il versionamento

3.2.5.1 Media commit a settimana

Media dei commit effettuati settimanalmente sulle diverse repository del gruppo. Utile per tenere traccia dell'impegno dedicato settimanalmente dal team sul progetto. Per questa metrica verranno utilizzati i risultati dei grafici generati automaticamente nella sezione *Insights/Commits* di GitHub.

3.3 Verifica

Il processo di verifica ha lo scopo di esaminare un prodotto al fine di accertare che le attività produttive non abbiano introdotto errori sullo stesso, fornendo una prova oggettiva della sua correttezza o nel caso contrario, segnalando le eventuali problematiche riscontrate. Le metriche ed i metodi per effettuare verifica sono ampiamente e dettagliatamente descritti nel *Piano di Qualifica v3.0.0*.

3.3.1 Analisi dei processi

La qualità dei processi è garantita dall'uso del ciclo di miglioramento continuo $PDCA_g$, descritto nell'appendice B. Con l'applicazione di questo metodo è possibile perseguire una migliore qualità dei processi, incluso quello di verifica; ciò porta ad ottenere una migliore qualità dei prodotti che ne derivano.
Per ottenere una migliore qualità, quindi, bisogna che un processo venga definito, controllato e valutato.

3.3.1.1 Analisi PDCA

Attraverso lo studio della rappresentazione grafica del $PDCA_g$ relativo alla fase di progetto in esame, verranno tratte delle considerazioni riguardo lo svolgimento dei processi. Da esso si possono identificare in modo visuale, quindi generico e non numericamente $quantificabile_g$, dei dati sulla fase in esame ed osservarne la tendenza:

- Errori di pianificazione, rappresentati da variazioni delle attività nello stato Plan;
- Velocità con cui il gruppo di lavoro porta avanti i processi tra i vari stati del ciclo PDCA.

3.3.2 Analisi dei documenti

Descrizione delle procedure da eseguire per verificare adeguatamente i documenti.

3.3.2.1 Controllo del periodo

Per verificare la presenza di errori di sintassi, di parole grammaticalmente corrette ma presenti in un contesto sbagliato e di periodi difficili da comprendere è necessario l'intervento umano. Per questo motivo ogni documento deve essere sottoposto a verifica da parte dei *Verificatori* incaricati, secondo la strategia walkthrough.

3.3.2.2 Rispetto delle Norme di Progetto

La verifica del rispetto delle norme descritte in *Norme di Progetto v3.0.0* è compito dei *Verificatori* incaricati. Non essendo possibile impiegare strumenti automatici per verificare la corretta applicazione di tutte le norme spetta quindi ai *Verificatori* sottoporre i documenti a verifica, secondo la strategia inspection.

3.3.2.3 Metriche per i prodotti documentali

In seguito ad una qualsiasi modifica di un documento, si devono calcolare tutti i relativi indici di quel specifico documento. Inoltre dovrà essere ricalcolata la media complessiva di tutti gli indici calcolati fino a quel momento del documento in questione. Di seguito verranno descritte in modo dettagliato tutte le metriche relative ai documenti adottate. Per una spiegazione dettagliata riguardo gli obbiettivi che il team si è imposto di raggiungere per le suddette metriche consultare il documento *Piano di Qualifica v3.0.0*.

3.3.2.3.1 Numero di errori grammaticali

Misura del numero di errori ortografici presenti all'interno di un documento. Per verificare la presenza di errori ortografici nella documentazione deve essere usato lo strumento di controllo ortografico offerto dall'editor *TexStudio* che integra i dizionari di OpenOffice per segnalare potenziali errori presenti nel testo.

3.3.2.3.2 Gunning fog index

Indice utilizzato per misurare la facilità di lettura e comprensione di un testo. Il numero risultante è un indicatore del numero di anni di educazione formale necessari al fine di leggere il testo con facilità.
L'indice di Gunning fog è calcolabile tramite la seguente formula:

$$0.4 * ((\frac{n^\circ\ parole}{n^\circ\ frasi}) + 100 * (\frac{n^\circ\ parole\ complesse}{n^\circ\ parole}))$$

Per ogni documento i *Verificatori* devono calcolare il Gunning fog index e se questo dovesse risultare troppo alto, dovrà essere eseguita la verifica del documento con l'obiettivo di ricercare frasi troppo prolisse o complesse. Il calcolo deve essere effettuato attraverso uno $script_g$ scritto in Perl che trasforma i documenti dal formato .pdf al formato .txt per effettuare un calcolo più preciso, eliminando le tabelle ma non il loro contenuto.

3.3.2.3.3 Indice di Gulpease

Utilizzato per misurare la leggibilità di un testo in lingua italiana.
L'indice di Gulpease è calcolabile tramite la seguente formula:

$$89 + \frac{(numero\ delle\ frasi) - 10 * (numero\ delle\ lettere)}{numero\ delle\ parole}$$

I risultati sono compresi tra 0 e 100, dove 0 indica la leggibilità più bassa e 100 la leggibilità più alta. In generale risulta che i testi con indice:

- **Inferiore a 80**: sono difficili da leggere per chi ha la licenza elementare;
- **Inferiore a 60**: sono difficili da leggere per chi ha la licenza media;
- **Inferiore a 40**: sono difficili da leggere per chi ha la licenza superiore.

Per ogni documento i *Verificatori* devono calcolare l'indice di Gulpease e se questo dovesse risultare troppo basso, dovrà essere eseguita la verifica del documento con l'obiettivo di ricercare frasi troppo prolisse o complesse. Il calcolo deve essere effettuato attraverso uno $script_g$ scritto in Perl che trasforma i documenti dal formato .pdf al formato .txt per effettuare un calcolo più preciso, eliminando le tabelle ma non il loro contenuto.

3.3.3 Analisi dei prodotti software

Abbiamo deciso di dotarci di uno strumento per il pre-commit che, prima di completare l'operazione di commit sulla repository, esegue l'analisi statica e l'analisi dinamica in locale

della nuova versione del codice; se l'esito di questi test dovesse risultare negativo allora il commit non andrà a buon fine, in caso contrario invece l'operazione avrà successo.

3.3.3.1 Analisi statica

Il processo di analisi statica consiste nello svolgere dei test che non richiedono l'esecuzione del sistema software ma effettuano controlli sul codice sorgente. Può essere svolta nei seguenti modi:

- **Walkthrough**: tecnica applicata quando non si sanno le tipologie di errori o problemi che si stanno cercando; è un'attività di lettura integrale e approfondita del testo o codice del prodotto, principalmente utilizzata durante le prime fasi del progetto in quanto permette una verifica più attenta e precisa dei prodotti. Ricade tra i compiti del *Verificatore*, che si occuperà anche di stilare una lista degli errori riscontrati per facilitare la discussione di eventuali modifiche e permettere di individuare gli errori più frequenti. La fase finale consiste nell'applicare e registrare le modifiche correttive approvate;
- **Inspection**: attività di analisi mirata di parti specifiche del prodotto documentale o software che sono ritenute sezioni critiche, ovvero con grande concentrazione, potenziale o effettiva, di errori o anomalie. La lista degli errori da controllare va compilata prima dell'inizio dell'attività, in quanto maturata dall'esperienza acquisita durante le precedenti attività di *Walkthrough*. Essendo limitata ad un'area specifica, risulta più veloce nell'esecuzione e nell'attuazione delle modifiche necessarie.

3.3.3.1.1 Strumenti di supporto

Ai fini di svolgere l'analisi statica del codice lo strumento usato è: *TSLint*$_g$ un *linter*$_g$ che mette a disposizione numerose regole e la possibilità di creare regole personalizzate. TSLint è inoltre integrabile in numerosi IDE e nella piattaforma TravisCI che permette di eseguire test automatici su repository GitHub.

3.3.3.2 Analisi dinamica

Il processo di analisi dinamica consiste nell'esecuzione di test di varia tipologia i quali richiedono l'esecuzione del sistema software o solamente di alcune sue componenti.
La consistenza dei test in esecuzioni ripetute è una condizione imperativa, dunque è necessario che essi siano *ripetibili*$_g$: un dato test eseguito in un ambiente specifico deve produrre, se fornito un determinato input, sempre gli stessi output. Al fine di garantire ciò, il team 7DOS ha scelto di basarsi sullo standard *ISO/IEC/IEEE 29119*$_g$[6] per quanto riguarda la pianificazione e documentazione dei test dinamici. Tale standard prevede di definire, per ciascun test o suite di test, una *specifica di test* composta di:

- **Specifica di progettazione:** definisce le funzionalità del prodotto da testare e le condizioni di test, ovvero l'ambiente di esecuzione e le pre-condizioni (particolari eventi o stati pregressi) necessarie al suo svolgimento;

[6] ISO/IEC/IEEE 29119 parte 3 sezione 7, IEEE 2013.

- **Specifica di caso:** definisce l'insieme degli input che si desidera testare, e l'insieme dei risultati attesi per ogni input per una o più funzionalità testate;
- **Specifica di procedura:** definisce l'ordine di esecuzione dei test (nel caso di una suite di test), la modalità di svolgimento, ovvero le azioni da compiere e gli input da inserire in modo ordinato, eventuali azioni necessarie per il raggiungimento delle pre-condizioni e la modalità di analisi dei risultati ottenuti.

Ogni test, per essere identificato in modo univoco, deve seguire la seguente sintassi:

T[Tipo]-[ID]

- **Tipo**: specifica il tipo del test e viene indicato con una delle seguenti lettere:
  - A: accettazione;
  - S: sistema;
  - I: integrazione;
  - U: unità.
- **ID**: indica il codice del test rispettando una struttura gerarchica.

Per ogni test viene indicato lo stato in cui si trova, con una delle seguenti sigle:

- **NI**: non implementato;
- I: implementato;
- NS: non superato;
- S: superato.

Le principali tipologie di test che è possibile eseguire durante l'analisi dinamica sono le seguenti:

- **Test di unità**: mirano alla verifica della parte più piccola di lavoro prodotta da un *Programmatore*, equivalente all'unità logica più piccola del prodotto, che può essere una singola classe, un metodo o funzione oppure un insieme di essi;
- **Test di integrazione**: mirano alla verifica di due o più unità già testate che vengono aggregate **incrementalmente** in una struttura più grande, rappresentando l'estensione logica del test di unità. In questo modo si può testare se il comportamento atteso dell'aggregato rispetta le previsioni. In caso negativo, è possibile che le singole unità contengano difetti residui da correggere oppure che i software utilizzati siano poco conosciuti e abbiano comportamenti inaspettati. La strategia di integrazione dei vari moduli scelta è quella *bottom-up*$_g$, la quale prevede che vengano testate per prime le procedure più a basso livello e passando, progressivamente, a procedure di più alto livello;
- **Test di sistema**: mirano alla verifica del prodotto software completo di tutte le sue componenti. Un software a cui vengono applicati questi tipi di test deve essere giunto ad una versione ritenuta definitiva. Essendo test significativi, sarà richiesta la supervisione dei *Verificatori* incaricati;

- **Test di regressione**: mirano ad eseguire nuovamente i test di unità e di integrazione su componenti software che hanno subito modifiche, in modo da controllare che i cambiamenti apportati non abbiano inserito nuovi errori sia nelle componenti modificate che nelle componenti non modificate e che prima non erano soggette ad errori;
- **Test di accettazione**: mirano al collaudo del prodotto software in presenza del *proponente*$_g$. Essi sono test finali il cui superamento comporta la validazione e il rilascio del prodotto.

3.3.3.2.1 Strumenti di supporto

Per l'analisi dinamica vengono usati Mocha e Chai. Il primo è un framework per l'esecuzione di test JavaScript che si basa su Node.js, il secondo è una libreria che contiene numerose interfacce per facilitare lo sviluppo dei test.

3.3.3.3 Metriche per i test

Di seguito verranno descritte in modo dettagliato tutte le metriche adottate relative ai test. Per una spiegazione dettagliata riguardo gli obiettivi che il team si è imposto di raggiungere per le suddette metriche consultare il *Piano di Qualifica v3.0.0.*

3.3.3.3.1 Media *Build*$_g$ *Travis*$_g$ a settimana

Media delle build Travis effettuate settimanalmente sulle diverse repository del gruppo. Utile per tenere traccia dell'impegno dedicato settimanalmente dal team sul progetto.

3.3.3.3.2 Percentuale di build Travis superate

Percentuale delle build Travis effettuate che hanno avuto esito positivo sul numero totale delle build effettuate. Utile per tenere traccia dei progressi raggiunti dal team.

3.3.3.3.3 Percentuale di test eseguiti

Indica la percentuale di test eseguiti sul totale di test da eseguire, serve al team di verificatori per monitorare lo svolgimento delle loro attività. Viene utilizzata la seguente formula:

$$PTE = \frac{TE}{TT} * 100$$

Dove:

- **PTE**: è il valore della metrica;
- **TE**: è il numero di test eseguiti;
- **TT**: è il numero di totale di test.

3.3.3.3.4 Percentuale test case passati

Indica la percentuale di test passati rispetto ai test totali.
La percentuale test case passati è calcolabile tramite la seguente formula:

$$PTP = \frac{TP}{NT} * 100$$

Dove:

- **PTP**: è il valore della metrica;
- **TP**: è il numero di test case passati;
- **NT**: è il numero di test case totale.

3.3.3.3.5 Percentuale test case falliti

Indica la percentuale di test falliti rispetto ai test totali.
La percentuale test case falliti è calcolabile tramite la seguente formula:

$$PTF = \frac{TF}{NT} * 100$$

Dove:

- **PTF**: è il valore della metrica;
- **TF**: è il numero di test case falliti;
- **NT**: è il numero di test case totale.

3.3.3.3.6 Tempo medio necessario al team per risolvere un errore

Indica la quantità di tempo media spesa dal team per risolvere una criticità, utile a comprendere l'impatto che l'introduzione di bug ha sui tempi di sviluppo. La formula usata per calcolarla è:

$$TMRE = \frac{TRE}{NE}$$

Dove:

- **TMRE**: è il valore della metrica;
- **TRE**: è il quantitativo di tempo speso a risolvere errori;
- **NE**: è il numero di errori risolti.

3.3.3.3.7 Efficienza nella progettazione dei test

Indica il tempo medio per la scrittura dei test. Un valore troppo basso potrebbe indicare la scrittura di test banali o poco efficaci. Al contrario un valore troppo alto potrebbe indicare

la scrittura di test troppo complessi che rischiano di contenere essi stessi errori. La formula usata per calcolare questa metrica è:

$$TDE = \frac{NTP}{TST}$$

Dove:

- **TDE**: è il valore della metrica;
- **NTP**: è il numero di test scritti;
- **TST**: è il quantitativo di tempo speso a scrivere test.

3.3.3.3.8 Percentuale di errori corretti

Indica la percentuale di criticità (bug, difetti in genere) risolte sul totale di quelle note. Se questa percentuale è troppo bassa, il team sarà costretto a fermare i processi di sviluppo per risolvere le criticità esistenti. Per il calcolo viene utilizzata la seguente formula:

$$PCR = \frac{CR}{CN} * 100$$

Dove:

- **PCR**: è il valore della metrica;
- **CR**: è il numero di criticità risolte;
- **CN**: è il numero di criticità note.

3.3.3.3.9 Code coverage

Per assicurarsi che la più grande porzione possibile di codice venga testata, si adotta questa metrica, che indica in percentuale, quanto del codice sorgente è soggetto a test. Più alta tale percentuale, minore il rischio che vi siano criticità non rilevate. Per il suo calcolo si farà uso degli strumenti automatici descritti nella sezione §2.2.2 di questo documento, ovvero TravisCI e Coveralls.

3.3.4 Analisi diagrammi UML

La verifica dei diagrammi UML prodotti è compito dei *Verificatori*, che devono accertarsi che rispettino lo standard UML e che siano semanticamente corretti.

3.3.5 Miglioramento del processo di verifica

Per migliorare e ottimizzare il processo di verifica i *Verificatori* devono riportare gli errori riscontrati più frequentemente in una *checklist*, al fine di rendere più efficiente ed efficace l'attività di verifica, prestando maggiore attenzione agli errori più comuni.

3.4 Validazione

Il processo di validazione deve verificare che il prodotto sia conforme a quanto pianificato e sia capace di gestire e minimizzare gli effetti degli errori.

3.4.1 Procedura di validazione

Per completare l'attività di validazione il prodotto deve seguire, nel giusto ordine, i seguenti passi:

1. Esecuzione dei test sul prodotto;
2. Verifica dei risultati ad opera dei *Verificatori*, che ne stendono un resoconto;
3. Lettura del resoconto da parte del *Responsabile di Progetto*. Se i test non danno risultati soddisfacenti, il *Responsabile* chiede di eseguire nuovamente i test e, se lo ritiene necessario, può aggiungere nuove indicazioni in merito. In caso di esito negativo, ripetere tutti i passi a partire dal punto 1;
4. Invio dei risultati alla proponente.

4 Processi organizzativi

4.1 Gestione del progetto

I punti principali dell'attività di gestione di progetto sono:

- L'istanziazione dei processi di progetto;
- La pianificazione e la gestione dei compiti e delle attività che compongono i processi. Obbiettivi di quest'attività sono:
  - Permettere l'analisi dei rischi;
  - Sviluppare una strategia di lavoro che faccia uso di *best practices*$_g$;
  - Permettere una stima dei costi delle attività di progetto.
- La verifica delle attività e la loro eventuale modifica nell'ottica del miglioramento. Per garantire che la verifica sia efficace deve:
  - Essere supportata da strumenti informatici che riducano il carico di verifica sulla persona;
  - Far uso di metriche numeriche ben definite.

4.1.1 Ruoli di progetto

Segue la lista dei ruoli di progetto che ogni membro dovrà ricoprire a rotazione. Le rotazioni andranno effettuate quando arrecheranno meno disagi alle attività di progetto e i ruoli saranno stabiliti casualmente con eventuali aggiustamenti per garantire che ogni membro possa assumere almeno una volta ogni ruolo.

4.1.1.1 Analista

L'*Analista* si occupa di determinare e descrivere in maniera formale i requisiti del prodotto, siano essi impliciti o espliciti. Deve:

- Comunicare con il committente per determinare i requisiti del prodotto e i requisiti di dominio;
- Produrre uno Studio di Fattibilità;
- Produrre l'Analisi dei Requisiti.

4.1.1.2 Progettista

Il *Progettista* fa uso dell'Analisi dei Requisiti per produrre un architettura che rispetti i requisiti definiti dall'*Analista*. Deve:

- Produrre un'architettura che faccia uso di best practices e che agevoli *manutenzione*$_g$ e verifica;
- Assicurarsi che l'architettura rispetti i requisiti definiti dall'*Analista*;

- Redigere la documentazione relativa all'architettura del prodotto e un manuale destinato al programmatore per l'implementazione dell'architettura progettata.

4.1.1.3 Programmatore

Il *Programmatore* ha come compito implementare l'architettura progettata dal *Progettista*. Deve:

- Scrivere codice che sia documentato e che rispetti le direttive del *Progettista*;
- Codificare i moduli necessari al testing del codice;
- Produrre il manuale utente.

4.1.1.4 Verificatore

Il *Verificatore* si occupa delle attività di verifica. Deve:

- Verificare la qualità dei prodotti delle varie attività;
- Verificare che le attività siano svolte rispettando le Norme di Progetto stabilite;
- Redigere il Piano di Qualifica.

4.1.1.5 Amministratore

L'*Amministratore* gestisce l'ambiente di lavoro del team. Deve:

- Determinare quali strumenti verranno usati dal team (sistema di versioning, *sistema di ticketing*$_g$, strumenti per la comunicazione, etc.);
- Garantisce l'operatività dei sistemi informatici a supporto delle attività di progetto.

4.1.1.6 Responsabile

Il *Responsabile* rappresenta il team nelle interazioni con il committente, si occupa di coordinamento interno, si assume la responsabilità delle scelte prese ed approva i prodotti dei processi. Deve:

- Gestire i processi di comunicazione interna ed esterna (incontri, comunicazioni con il committente, riunioni del team);
- Approvare i prodotti risultanti dei processi (documenti, altro);
- Coordinare il gruppo assegnando attività ai vari membri e scadenze e gestendo i rischi.

4.1.2 Sistema di ticketing

4.1.2.1 Gestione dei task

Vengono distinti 2 tipi di task:

- Pro-attivi: creati e assegnati dal *Responsabile di Progetto*, organizzano lo svolgimento delle attività e la verifica;
- Reattivi: creati e assegnati dai *Verificatori*, organizzano la correzione o il *miglioramento*$_g$ dei prodotti delle attività.

Qualora l'assegnatario di un task dovesse realizzare tale task non è completabile entro la data di fine, o se ci dovessero essere altri problemi con il task, avrà la responsabilità di segnalare il problema al *Responsabile di Progetto* che sarà tenuto a ripianificare la risoluzione del task.

4.1.2.2 Norme sui task

Il nome del task deve essere nella forma [attività] nome. Il nome deve essere descrittivo del task stesso. Ogni task dovrà avere almeno un assegnatario.

4.1.2.3 Strumenti di supporto

4.1.2.3.1 nTask

Per la gestione del sistema di ticketing viene utilizzato nTask. La piattaforma permette al *Responsabile di Progetto* di creare *task*$_g$ e di assegnarli ai membri del gruppo. Ogni task è caratterizzato dalla data di inizio, data di fine, una descrizione ed una eventuale checklist. Esso permette di avere diverse visualizzazioni dei task assegnati: lista testuale, griglia e calendario. Abbiamo deciso di utilizzare questo strumento, data la sua comoda board view, la possibilità di creare più feature anche nella versione gratuita e di avere uno strumento di *risk tracking*$_g$ integrato.

4.1.2.4 Meccanismi di controllo

Il sistema di ticketing adottato, permette di suddividere e controllare le varie attività. In particolare, è possibile visualizzare:

- **Calendario attività**: la data di fine delle varie attività è indicata in un calendario e gestita automaticamente dal sistema di ticketing;
- **Dettaglio attività**: ogni attività include diverse informazioni, quali personale incaricato, stato dell'attività e data di fine prevista.

4.1.2.4.1 Controllo dei ritardi

Ogni attività, assegnata a uno o più componenti del gruppo, deve essere svolta entro una data prestabilita. In questo modo è possibile pianificare in modo chiaro ed efficace l'insieme delle varie attività. Il mancato completamento entro la data prevista comporta un ritardo. In questo caso, abbiamo deciso di chiudere la task tenendo conto delle ore effettive di lavoro fino a quel momento. Successivamente, verrà aperta una nuova task uguale alla precedente contrassegnata da una "R", che simboleggia un ritardo nel completamento. Così facendo, le

ore aggiuntive necessarie possono, in ogni caso, essere segnate nella nuova task. Alla fine, si avrà un totale delle ore impiegate, le quali verranno rendicontate.

4.1.2.5 Meccanismi di rendicontazione

Il sistema di ticketing adottato, permette ad ogni membro di rendicontare le ore spese per una specifica attività. In particolare, è possibile visualizzare:

- Ore di lavoro complessive per un attività;
- Ore di lavoro complessive per un ruolo.

4.1.3 Metriche per la gestione del progetto

4.1.3.1 *Schedule Variance (SV)*$_g$

Metrica che indica se il progetto è in linea, in anticipo o in ritardo rispetto alle attività pianificate nella *baseline*$_g$. Essa è un indicatore di efficacia.
Calcolabile attraverso la seguente formula:

$$SV = BCWP - BCWS$$

Dove:

- **SV**: corrisponde al valore della metrica;
- **BCWP**: corrisponde al valore, in giorni od euro, prodotto dalle attività del progetto realizzate fino alla data corrente, od in altre parole la somma complessiva del valore di tutte le componenti completate fino ad ora.
- **BCWS**: corrisponde al numero di giorni od euro pianificati inizialmente per realizzare le attività fino alla data corrente.

Se SV $>$ 0 significa che si sta procedendo più velocemente rispetto a quanto pianificato. Viceversa, se SV $<$ 0 allora il gruppo è in ritardo.

4.1.3.2 *Budget Variance (BV)*$_g$

Metrica che indica se la spesa complessiva fino a quel momento è maggiore o minore rispetto a quanto pianificato. Essa è un indicatore con valore contabile e finanziario.
Calcolabile attraverso la seguente formula:

$$BV = \frac{BCWS - ACWP}{BCWS}$$

- **BV**: corrisponde al valore della metrica;
- **BCWS**: corrisponde al costo pianificato in giorni od euro per realizzare le attività del progetto fino alla data corrente;
- **ACWP**: corrisponde al costo, in giorni od euro, effettivamente sostenuto fino alla data corrente.

Se BV > 0 il budget complessivo sta diminuendo con velocità minore rispetto a quanto pianificato. Viceversa se BV < 0.

4.1.3.3 Numero rischi non previsti

Indica la quantità di rischi rilevati durante lo svolgimento delle attività, non preventivati all'interno dell'analisi dei rischi durante l'attuale fase del progetto.

4.1.3.4 Indisponibilità servizi esterni

Numero complessivo di giorni in cui uno specifico servizio esterno, tra tutti quelli utilizzati, è rimasto indisponibile per la maggior parte del tempo rendendone quindi impossibile l'utilizzo. Sarà necessario utilizzare un contatore separato per ogni servizio utilizzato.

4.2 Gestione delle comunicazioni

4.2.1 Comunicazioni interne al gruppo

Le comunicazioni interne al gruppo sono gestite principalmente tramite due servizi di messaggistica Telegram e Discord.

4.2.1.1 Norme su Discord

Discord viene impiegato dal team per effettuare comunicazioni di urgenza limitata. Esso viene spesso utilizzato per effettuare riunioni a distanza tramite i suoi canali vocali. È possibile organizzare le conversazioni creando vari canali vocali e testuali associati ad uno scopo o ad alcuni ruoli del progetto, e raggruppandoli in sezioni. Di seguito verrà riportata una breve descrizione dei canali attualmente presenti, tuttavia è importante sottolineare che molti canali potrebbero essere aggiunti o rimossi nel tempo e che quindi questa lista potrebbe cambiare in futuro.

- **Text channels**: sezione che contiene tutti i canali testuali principali del server. Di seguito verranno riportati tutti i canali contenuti in questa sezione:
  - **General**: canale testuale generale in cui effettuare comunicazioni poco urgenti destinate a tutti i membri del team;
  - **Deadlines**: canale testuale in cui vengono scritte tutte le deadline imminenti in modo da ricordarle a tutti i membri del team;
  - **Impegni-personali**: canale testuale in cui vengono scritti i periodi di assenza dei membri del team, per garantire una migliore pianificazione.
- **Voice channels**: sezione che contiene tutti i canali vocali principali del server. Di seguito verranno riportati tutti i canali contenuti in questa sezione:
  - **General**: canale vocale generico utilizzato principalmente per effettuare riunioni che coinvolgono tutti i membri del team;

  - **Redazione**: canale vocale utilizzato principalmente per effettuare riunioni tra i *Redattori* del team;
  - **Verifica**: canale vocale utilizzato principalmente per effettuare riunioni tra i *Verificatori* del team.

- **Notification center**: sezione che contiene tutti i canali testuali collegati ad un *Webhook*$_g$ per ricevere notifiche a particolari piattaforme esterne.
  - **Github-notifications**: canale testuale in cui verranno comunicate da un bot tutte le notifiche provenienti da Github;
  - **Gmail-notifications**: canale testuale in cui verranno comunicate da un bot tutte le notifiche relative all'indirizzo email del gruppo;
  - **Travis-build**: canale testuale in cui verranno comunicate da un bot tutte le notifiche relative alle build effettuate con TravisCI.
- **Documenti**: sezione contenente un canale testuale per ogni documento redatto dal team in modo tale da permettere un'efficace comunicazione tra i *Redattori* di quello specifico documento.

4.2.1.2 Norme su Telegram

Telegram viene utilizzato principalmente per effettuare comunicazioni urgenti ed organizzare riunioni che coinvolgono tutti i membri del team e necessitano di essere effettuate in tempi brevi. Di conseguenza è sconsigliato utilizzare questo servizio per effettuare comunicazioni che non appartengano a nessuna delle categorie precedentemente citate.

4.2.2 Comunicazioni esterne

Le comunicazioni esterne sono compito del *Responsabile di Progetto*, che utilizzerà l'indirizzo e-mail del gruppo 7dos.swe@gmail.com facendo così apparire una notifica nel server Discord del team una volta inviata la mail.

4.2.3 Strumenti di supporto

4.2.3.1 Discord

Discord, è un software di messaggistica istantanea e di VoIP che abbiamo scelto di utilizzare per le comunicazioni interne al gruppo, come spiegato al punto 4.2.1. Permette la creazione gratuita di un server nel quale comunicare con gli altri membri mediante canali testuali e vocali. Permette inoltre la creazione di un numero illimitato di Webhooks, utili per integrare molti altri strumenti. Il maggior vantaggio rispetto a Slack sta nell'essere un software con tutte le funzionalità di base disponibili gratuitamente, rendendo molto vantaggioso avere la possibilità di trovarsi a discutere con tutti gli altri membri del gruppo, cosa non possibile nella versione gratuita di Slack che offre solo chiamate 1-1.

4.2.3.2 Telegram

Servizio di messaggistica istantanea basato su cloud, il quale permette di stabilire conversazioni testuali tra due o più utenti.

4.3 Gestione degli incontri

4.3.1 Incontri interni

Il *Responsabile di Progetto* ha il compito di decidere il giorno e la data di un incontro, discutendone con gli altri membri del team. Per comodità, abbiamo scelto il martedì come giorno di riferimento da proporre inizialmente ai membri del team.
Una volta raggiunto un accordo su un giorno con un numero consistente di partecipanti (almeno 3), tutti i membri che hanno accettato dovranno presentarsi in orario nel luogo prestabilito per la riunione, comunicando eventuali ritardi o impegni insormontabili. Ad ogni incontro, tutti devono partecipare attivamente alla discussione offrendo la loro opinione sui punti da decidere e discutere, mentre un membro del team scelto dal *Responsabile* prenderà appunti sui contenuti da inserire appena possibile in un verbale.
Se non è possibile trovare un giorno in cui effettuare una riunione in un luogo fisico, si tenterà di scegliere un giorno in cui trovarsi a discutere su Discord.

4.3.2 Incontri esterni

Gli incontri esterni saranno anch'essi organizzati dal *Responsabile di Progetto*, che si metterà in contatto con il committente o il proponente tramite email come descritto al punto 4.1.2 per accordarsi con essi sulla data, l'ora e il luogo dell'incontro. Una volta che questi ultimi saranno stati decisi e confermati da entrambe le parti, il *Responsabile* avviserà i membri utilizzando Discord. In caso di impossibilità da una delle due parti ad organizzare un incontro fisico, si considererà l'opzione di un incontro utilizzando un servizio VoIP come Skype, Hangout o simili.

4.4 Formazione

La formazione ai fini del progetto è responsabilità di ogni singolo membro del gruppo. Qualora ai fini del progetto un membro del gruppo dovesse formarsi autonomamente su un argomento sconosciuto agli altri, sarà sua responsabilità informare il gruppo e se ritenuto opportuno formarlo secondo una delle seguenti modalità:

- Redigere un documento a scopo introduttivo che illustri le tematiche principali ed elenchi le risorse usate per la formazione. Tale modalità è da preferirsi alla successiva che verrà adottata se le circostanze dovessero ostacolare la redazione del documento;
- Formare il gruppo tramite un incontro a scopo formativo.

A ISO/IEC 15504(SPICE)

Per ogni *processo*$_g$ lo standard ISO/IEC 15504 definisce una scala di maturità a cinque livelli (più il livello base, detto "livello 0"), riportati di seguito:

- **Livello 0 - Incomplete Process**: il processo riporta *performance*$_g$ e risultati incompleti, inoltre è gestito in modo caotico;
- **Livello 1 - Performed Process**: il processo raggiunge i risultati attesi ma viene eseguito in modo non controllato. Gli attributi di tale processo sono:
  - **1.1 - Process Performance**: capacità di un processo di raggiungere gli obiettivi definiti.
- **Livello 2 - Managed Process**: il processo è pianificato e tracciato secondo standard prefissati, dunque il suo prodotto è controllato, manutenuto e soddisfa determinati criteri di qualità. Gli attributi di tale processo sono:
  - **2.1 - Performance Management**: capacità di un processo di identificare gli obiettivi e di definire, monitorare e modificare le sue performance;
  - **2.2 - Work Product Management**: capacità di un processo di identificare, elaborare, documentare e controllare i propri risultati.
- **Livello 3 - Established Process**: il processo possiede specifici standard organizzativi che includono linee guida personalizzate, il tutto è consolidato tramite una politica di feedback del prodotto. Gli attributi di tale processo sono:
  - **3.1 - Process Definition**: il processo specifico si basa su processi standard, individuando e incorporando caratteristiche fondamentali di questi;
  - **3.2 - Process Deployment**: sono stati definiti ed assegnati dei ruoli a ciascun membro del team, ogni risorsa necessaria per l'esecuzione del processo è disponibile ed utilizzabile.
- **Livello 4 - Predictable Process**: il processo è quantitativamente misurato e statisticamente analizzato per permettere di prendere decisioni oggettive e per assicurare che le prestazioni rimangano all'interno di limiti definiti. Gli obiettivi sono, di conseguenza, supportati in maniera consistente. Gli attributi di tale processo sono:
  - **4.1 - Process Measurement**: i risultati di misurazione dei processi vengono utilizzati per garantire che le prestazioni del processo supportino il raggiungimento degli obiettivi determinati;
  - **4.2 - Process Control**: il processo è gestito quantitativamente in modo da renderlo stabile, capace e prevedibile entro limiti definiti.
- **Livello 5 - Optimizing Process**: il processo è in continuo miglioramento per raggiungere adeguatamente gli obiettivi prefissati. Gli attributi di tale processo sono:
  - **5.1 - Process Innovation**: gli obiettivi di miglioramento del processo supportano gli obiettivi aziendali rilevanti;

  - **5.2 - Process Optimization**: le modifiche alla definizione, gestione e le prestazioni del processo si traducono in un impatto efficace per il raggiungimento degli obiettivi.

Lo standard SPICE offre una scala di valutazione per ogni processo, in modo da misurare il livello di raggiungimento degli stessi:

- **N - Not Achieved**: 0 - 15%;
- **P - Partially Achieved**: >15% - 50%;
- **L - Largely Achieved**: >50% - 85%;
- **F - Fully Achieved**: >85% - 100%.

B Ciclo di miglioramento continuo (PDCA)

Ogni processo, per perseguire una migliore qualità, deve basarsi sul ciclo di miglioramento continuo (PDCA, *Plan-Do-Check-Act)*:

- **Plan**: viene definito un piano che, sulla base di problemi e obiettivi, pianifica compiti da svolgere, assegna delle responsabilità, studia il caso, analizza le ragioni delle criticità e ne definisce le azioni correttive;
- **Do**: vengono implementate le attività secondo quanto stabilito durante la fase di Plan;
- **Check**: viene verificato l'esito delle azioni di miglioramento rispetto alle attese;
- **Act**: vengono effettuate le correzioni per migliorare le criticità rilevate e vengono standardizzate le attività eseguite correttamente.

C ISO/IEC 25010 (SQuaRE)

C.1 Functional Suitability

Questa caratteristica esprime il grado di soddisfacimento dei requisiti espliciti ed impliciti da parte di un prodotto o servizio, quando utilizzato sotto determinate condizioni.

Sotto-caratteristiche notevoli:

- **Functional Completeness**: esprime il grado con cui l'insieme di funzioni copre i compiti specificati e gli obiettivi dell'utente;
- **Functional Correctness**: esprime il grado con cui il prodotto restituisce risultati corretti, entro il livello di precisione desiderato.

C.2 Performance Efficiency

Questa caratteristica esprime le prestazioni relative al sistema come, quantità di risorse utilizzate per eseguire una determinata funzionalità del sistema sotto specifiche condizioni.

Sotto-caratteristiche notevoli:

- **Time Behaviour**: esprime il grado con cui i tempi di risposta ed elaborazione e i volumi di produzione di un prodotto o sistema, durante l'esecuzione delle sue funzionalità, rispettano i requisiti;
- **Resource Utilization**: esprime il grado con cui il numero e tipo di risorse utilizzate da un prodotto o sistema, durante l'esecuzione delle sue funzionalità, rispetta i requisiti.

C.3 Usability

Questa caratteristica esprime il grado con cui un prodotto o sistema può essere usato da un determinato utente per raggiungere determinati scopi con efficacia, efficienza e soddisfazione in uno specifico contesto d'uso.

Sotto-caratteristiche notevoli:

- **Learnability**: esprime il grado con cui determinati utenti sono in grado di imparare ad utilizzare il prodotto o sistema efficacemente, efficientemente, con sicurezza da rischi e soddisfazione in un dato contesto d'uso;
- **Operability**: esprime il grado con cui un prodotto o sistema ha attributi che lo rendono facile da operare e controllare;
- **User Error Protection**: esprime il grado di efficacia ed efficienza con cui un sistema protegge gli utenti dal commettere errori.

C.4 Reliability

Questa caratteristica esprime il grado con cui un sistema, prodotto o componente esegue determinate funzioni sotto specifiche condizioni per un dato periodo di tempo.

Sotto-caratteristiche notevoli:

- **Maturity**: esprime il grado con cui un sistema, prodotto o componente raggiunge i requisiti di affidabilità in normali condizioni operative;
- **Fault Tolerance**: esprime il grado con cui un sistema, prodotto o componente opera come previsto nonostante la presenza di malfunzionamenti hardware o software.

C.5 Maintainability

Questa caratteristica esprime il grado di efficacia ed efficienza con cui un prodotto, o componente può essere modificato per migliorarlo, correggerlo o adattarlo a dei cambiamenti all'ambiente.

Sotto-caratteristiche notevoli

- **Modularity**: esprime il grado di scomposizione del sistema in parti minimali tali che un cambiamento ad una specifica componente ha il minimo impatto su tutte le altre componenti;
- **Analyzability**: esprime il grado di efficacia ed efficienza con cui è possibile analizzare l'impatto nel sistema di uno specifico cambiamento ad una o più delle sue parti, ai fini di rilevare eventuali casi di fallimento;
- **Modifiability**: esprime il grado con cui un prodotto o sistema può essere modificato efficacemente ed efficientemente senza introdurre difetti che ne possano intaccare la qualità complessiva;
- **Testability**: esprime il grado di efficacia ed efficienza con cui è possibile stabilire ed eseguire dei test per valutare la qualità del sistema.